NASA-CR-194740

031155-1-T

A new conformal absorbing boundary condition for finite element meshes and parallelization of FEMATS

A. Chatterjee, J. L. Volakis, J. Nguyen,
M. Nurnberger and D. Ross

National Aeronautics
and Space Administration
Ames Research Center
Moffett Field CA 94035

December 1993

THE UNIVERSITY OF MICHIGAN
Radiation Laboratory
Department of Electrical Engineering
and Computer Science
Ann Arbor, Michigan 48109-2122
USA

Report #031155-1-T

NASA Ames Grant NAG 2-866

PROGRESS REPORT

Grant Title:	Development and Parallelization of the Finite Element Method with Mixed Termination Schemes
Report Title:	A new conformal absorbing boundary conditions for finite element meshes and parallelization of FEMATS
Report Authors:	A. Chatterjee, J. L. Volakis, J. Nguyen, M. Nurnberger and D. Ross
Primary University Collaborator:	John L. Volakis Volakis@um.cc.umich.edu Telephone: (313) 764-0500
Primary NASA-Ames Collaborator:	Alex Woo woo@ra-next.arc.nasa.gov Telephone: (415) 604-6010
University Address:	Radiation Laboratory Department of Electrical Engineering and Computer Science The University of Michigan Ann Arbor MI 48109-2122
Date:	December 1993

Funds for the support of this study have been allocated in part by the NASA-Ames Research Center, Moffett Field, California, under Grant No. NAG 2-866.

TABLE OF CONTENTS

Preface

This report describes some of our progress toward the development and parallelization of an improved version of the finite element code FEMATS. This is a finite element code for computing the scattering by arbitrarily shaped three dimensional surfaces composite scatterers.We have been working on the following tasks:

1. New ABCs for truncating the finite element mesh
2. Mixed mesh termination schemes
3. Hierarchical elements and multigridding
4. Parallelization
5. Various modeling enhancements(antenna feeds, anisotropy, higher order GIBC)

So far (approx. 5 months after the start date) we have concentrated on tasks 1, 3 and 5. Most of this report is devoted to task 1 and task 4. However, progress have been made toward task 2 and 5, although not yet formally written.

- The first section describes the new absorbing boundary conditions and their implementation. As pointed out in this section, care must be exercised in the implementation of the code to yield a symmetric matrix. Much effort was devoted on this aspect before the final implementation was carried out. Obviously, a good portion of our effort was devoted to the validation of the new ABCs and the new version of FEMATS. Toward this purpose, the RCS of several EMCC benchmark targets is included in this section. These include the rectangular inlet, circular inlet, metallic plates and the dielectric (glass) plate. In the case of the rectangular plate, it is important to note that substantial improvement was achieved on using the new ABCs. It remains to consider the "almond" target and the dielectric cylinder with embedded wires and our intent was to have included data for at least one of these targets. Unfortunately, hardware difficulties and scheduled shutdowns of the University's MPP did not allow us to complete runs for either the almond or the dielectric cylinder before the end of the year. Our findings clearly demonstrate that sufficiently accurate results can be obtained by placing the new ABC only 0.3 wavelengths away from the target and on a boundary which "hugs" the outline of the structure. The computational and memory savings are indeed substantial when such a conformal ABC is employed in FEMATS, and this is quatitatively depicted in the attached figures.
- The second section of the report describes the approach used to parallelize FEMATS on the Intel iPSC/860. This parallelization is, of course, pertaining to the earlier version of FEMATS which does not incorporate the new conformal ABCs. Both the solver and the matrix generation sections of the code were parallelized. In addition, preliminary results are presented on the code's performance on the iPSC/860. It should be noted that the code was originally parallelized on the KSR1 and many of the runs reported in this report have been executed on the University's 64-processor KSR1.

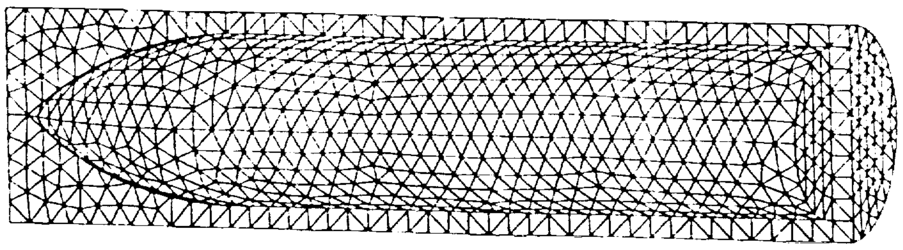
- The third section of the report is the code's users manual. This manual describes the input data format, preprocessor and the postprocessor. It is assumed that the user will be familiar with some three dimensional mesh generator. Currently, the code is interfaced with the commercial mesh generator SDRC I-DEAS. However, very simple modification are needed in the preprocessor in order to interface with other packages.
- Finally, Appendix A of this report describes a preprocessor which can be used to automatically find the metallic surfaces and outer boundary surfaces of the mesh. This algorithm will simplify the preprocessing of the mesh data but due to time restrictions, it has not been incorporated into final version of the FEMATS code.

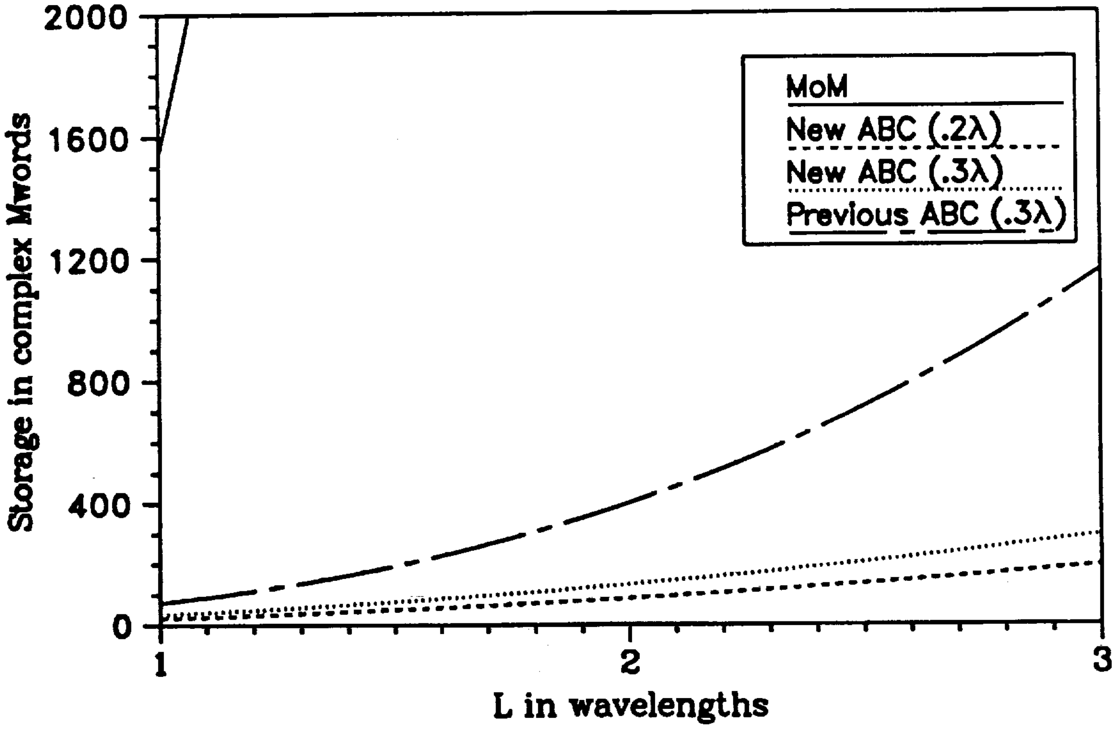
Future Work

We are looking at a variety of tasks aimed at improving the efficiency, accuracy anduser friendliness of the FEMATS code. Among these are:

1. The incorporation of graph tools into the preprocessor to eliminate communication difficulties between the mesh generator and the FEMATS electromagnetics solver.
2. Parallelization of the FEMATS version using the new ABCs
3. Additional validation
4. Use of higher order/hierarchical elements at the outer layer of the mesh to accommodate higher order ABCs
5. Improved solvers
6. Mixing of tetrahedral and quadrilaterals to reduce the degrees of freedom
7. Incorporation of a hybrid high frequency/finite element formulation
8. Antenna applications
9. Validation for targets greater than 5 wavelengths in length
10. Validation of FEMATS for anisotropic material simulations.

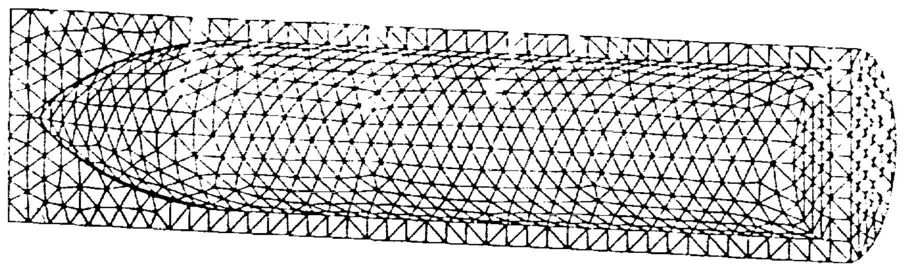
and so on.

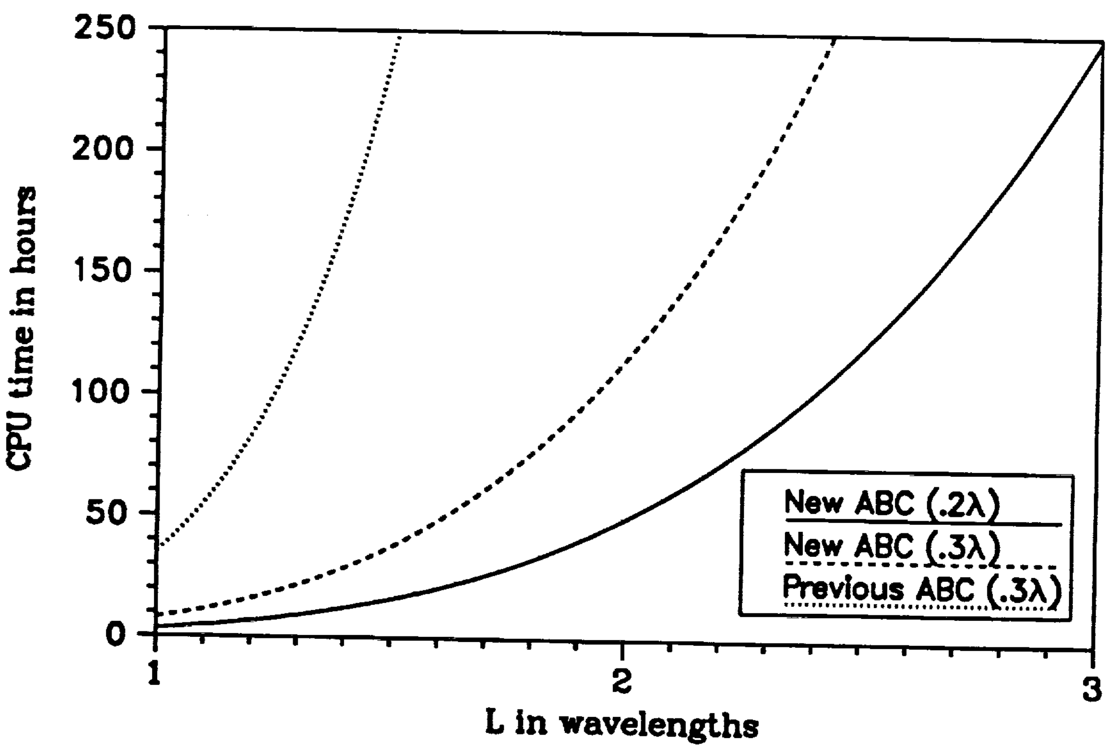




V_1 = volume of cylinder
V_2 = volume enclosed by ABC boundary
Number of unknowns $= 20,000(V_2 - V_1) = N$
Storage $= 27N$

Projected storage requirements of the envisioned finite element code using the new proposed ABCs for truncating the mesh conformal to the structure. The curves correspond to a metallic cylinder of radius $2.5L$ and length $10L$. For the previous ABCs the mesh is truncated on a rectangular parallelepiped as required by the physics of this ABC. Note that the corresponding Moment Method (MoM) curve is off scale.





N = # of unknowns (see Figure 6)
CPU times on KSR1 = $(1.37 \times 10^{-6} \times N \times \frac{N}{80})$ sec
$\frac{N}{80}$ = convergence rate

Projected CPU times of the envisioned finite element code using the new proposed ABCs placed at 0.3λ and 0.2λ away from the surface of a cylindrical structure. Also shown is the corresponding CPU curve using the previous ABC enforced on a parallelepiped enclosing the cylinder.

New conformal absorbing boundary conditions for the FEMATS code

1 Introduction

Since we are dealing with large targets having arbitrary shape, a spherical mesh termination boundary is not as attractive in terms of storage and computational cost. This is especially true for long and thin geometries where a sphere is the least economical shape of mesh termination, in terms of the number of unknowns. The ideal situation would be to enclose the scatterer inside a mesh termination boundary which is of the same shape as the scattering body (see Figure 1). If boundary conditions could be derived for such conformal mesh truncation surfaces, the volume to be meshed and the corresponding computing cost would then be minimized. However, available three dimensional ABCs for the vector wave equation as derived by Peterson[1] and Webb and Kanellopoulos[2] are only suited for application on spherical mesh terminations.

Our goal, therefore, is to derive new vector ABCs for three dimensional analysis which can be applied on a surface conformal to the structure of interest. We begin with a modified Wilcox expansion whose leading order term recovers the geometrical optics fields and thus, given the appropriate principal curvatures, the resulting ABC completely absorbs all geometrical optics fields from the surface. We then proceed to derive the first and second order absorbing boundary conditions in terms of the principal curvatures of the surface on which they are employed. We also introduce an approximation to make the absorbing boundary condition contribution symmetric. In the next step, we incorporate these boundary conditions into the finite element

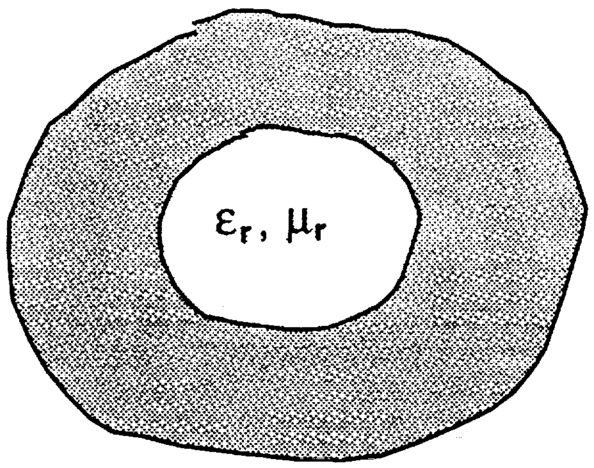


Figure 1: Scatterer enclosed in conformal mesh termination boundary

equations and express them in a readily implementable form. We also comment on the symmetry of the system for doubly curved surfaces. In the last section, we examine the performance of these ABCs - in terms of computational cost - when applied on mesh termination surfaces conformal to the scattering object.

2 Formulation

It is known that the electric field in a homogeneous region of space is governed by the vector wave equation

$$\nabla \times \nabla \times \mathbf{E} - k_o^2 \mathbf{E} = 0 \tag{1}$$

where k_o is the free-space wave number. We assume that the field has a well-defined phase front in the region under consideration. Since we are concerned only with local behavior, we can assume that the phase fronts can be treated as parallel regions. Consequently, the surface describing the phase fronts can be specified by a net of coordinate curves denoted by t_1 and t_2 and a third variable n denotes the coordinate along the normal to the phase front. The point of observation in the Dupin coordinate system[3] can now be defined as

$$\mathbf{x} = n\hat{\mathbf{n}} + \mathbf{x}_o(t_1, t_2) \tag{2}$$

where $\hat{\mathbf{n}}$ is the unit normal and $\mathbf{x}_o(t_1, t_2)$ denotes the surface of the reference phase front. The curl of a vector in the above coordinate system is given by

$$\nabla \times \mathbf{E} = \nabla_T \times \mathbf{E} + \hat{\mathbf{n}} \times \frac{\partial \mathbf{E}}{\partial n} \tag{3}$$

where $\nabla_T \times \mathbf{E}$ is called the surface curl involving only the tangential derivatives and is defined as [4]

$$\nabla_T \times \mathbf{E} = -\hat{\mathbf{n}} \times \nabla E_n + \hat{\mathbf{t}}_2 \kappa_1 E_{t_1} - \hat{\mathbf{t}}_1 \kappa_2 E_{t_2} + \hat{\mathbf{n}} \nabla \cdot (\mathbf{E} \times \hat{\mathbf{n}}) \tag{4}$$

In (4), κ_1 and κ_2 denote the principal curvatures of the surface under consideration, E_{t_1}, E_{t_2} are the tangential components and E_n is the normal component of the electric field on the surface. The principal curvature of a surface is defined as[3]

$$\kappa_1 = \frac{1}{R_1} = -\frac{1}{h_1}\frac{\partial h_1}{\partial n} \tag{5}$$

$$\kappa_2 = \frac{1}{R_2} = -\frac{1}{h_2}\frac{\partial h_2}{\partial n} \tag{6}$$

where h_1, h_2 are the metric coefficients and R_1, R_2 are the principal radii of curvature.

Using the aforementioned coordinates, the Wilcox expansion for a vector radiating function can now be generalized to read

$$\mathbf{E}(n, t_1, t_2) = \frac{e^{-jk_o n}}{4\pi\sqrt{R_1 R_2}} \sum_{p=0}^{\infty} \frac{\mathbf{E}_p(t_1, t_2)}{\left(\sqrt{R_1 R_2}\right)^p} \tag{7}$$

where $R_i = \rho_i + n$, $i = 1, 2$ and ρ_i is the principal radius of curvature associated with the outgoing wavefront at the target. The lowest-order term in (7) represents the geometrical optics spread factor for a doubly curved wavefront and reduces to the standard Wilcox expansion[5] for a spherical wave. Moreover, (7) can be differentiated term by term any number of times and the resulting series converges absolutely and uniformly[5].

2.1 Unsymmetric ABCs

In the 3D finite element implementation using vector basis functions and the electric field as the working variable, we need to relate the tangential component of the magnetic field in terms of the electric field at any surface discontinuity. Therefore, our next task is to derive a relation between $\hat{\mathbf{n}} \times \nabla\times\mathbf{E}$ (i.e., $\hat{\mathbf{n}} \times \mathbf{H}$ where $\mathbf{H}$ is the magnetic field) and the tangential components of the electric field on the surface. Taking the curl of the electric field expansion given by (7) and crossing it with the normal vector, we have

$$\hat{\mathbf{n}} \times \nabla\times\mathbf{E} = \frac{e^{-jk_o n}}{4\pi} \sum_{p=0}^{\infty} \left[\frac{(jk_o + \kappa_m - \overline{\boldsymbol{\eta}}\cdot)\mathbf{E}_{pt}}{u^{p+1}} + \frac{\nabla_t E_{pn} + p\kappa_m \mathbf{E}_{pt}}{u^{p+1}} \right] \quad (8)$$

where $u = \sqrt{R_1 R_2}$ and

$$\begin{aligned} \nabla_t E_n &= -(\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \nabla) E_n \\ \kappa_m &= \frac{\kappa_1 + \kappa_2}{2} \\ \overline{\boldsymbol{\eta}} &= \kappa_1 \hat{\mathbf{t}}_1 \hat{\mathbf{t}}_1 + \kappa_2 \hat{\mathbf{t}}_2 \hat{\mathbf{t}}_2 \end{aligned}$$

Considering that E_{0n} is zero due to the divergenceless condition[5] and simplifying, we obtain the first order absorbing boundary condition

$$\begin{aligned} \hat{\mathbf{n}} \times \nabla\times\mathbf{E} - (jk_o + \kappa_m - \overline{\boldsymbol{\eta}}\cdot)\mathbf{E}_t &= \frac{e^{-jk_o n}}{4\pi} \sum_{p=1}^{\infty} \frac{\nabla_t E_{pn} + p\kappa_m \mathbf{E}_{pt}}{u^{p+1}} & (9) \\ \text{or,}\ \hat{\mathbf{n}} \times \nabla\times\mathbf{E} - (jk_o + \kappa_m - \overline{\boldsymbol{\eta}}\cdot)\mathbf{E}_t &= 0 \quad + O\left(n^{-3}\right) & (10) \end{aligned}$$

for a conformal outer boundary. Not surprisingly, this is the impedance boundary condition for curved surfaces as derived by Rytov[6]. It should be noted that in the above equation, $\nabla_t E_n$ and κ_m are each proportional to n^{-1}. Therefore, the leading order behavior of (10) is $O(n^{-3})$, i.e., only the first two terms of (7) are exactly satisfied by (10). If the scattered field contains higher order terms, application of (10) will give rise to non-physical reflections back into the computational domain. In order to reduce these spurious reflections, we need to either shift the mesh truncation boundary farther away from the scatterer or employ higher order boundary conditions which satisfy higher order terms of (7).

To reduce the order of the residual error further, we consider the tangential components of the curl of (9). This yields

$$\hat{\mathbf{n}} \times \nabla\times \left[\hat{\mathbf{n}} \times \nabla\times\mathbf{E} - (jk_o + \kappa_m - \overline{\boldsymbol{\eta}}\cdot)\mathbf{E}_t\right] =$$

$$\frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}\left[\left\{jk_o+(p+3)\kappa_m-\frac{\kappa_g}{\kappa_m}\right\}\frac{\nabla_t E_{pn}+p\kappa_m\mathbf{E}_{pt}}{u^{p+1}}\right.$$
$$\left.-\left(2\kappa_m-\frac{\kappa_g}{\kappa_m}\right)\frac{\nabla_t E_{pn}}{u^{p+1}}-\frac{p\kappa_m\overline{\boldsymbol{\eta}}\cdot\mathbf{E}_{pt}}{u^{p+1}}\right] \qquad (11)$$

where $\kappa_g = \kappa_1\kappa_2$ is the Gaussian curvature. Using the result derived in (9) and simplifying, (11) reduces to

$$\hat{\mathbf{n}}\times\nabla\times\left[\hat{\mathbf{n}}\times\nabla\times\mathbf{E}-(jk_o+\kappa_m-\overline{\boldsymbol{\eta}}\cdot)\,\mathbf{E}_t\right]=$$
$$\frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}\left[\left\{jk_o+(p+3)\kappa_m-\frac{\kappa_g}{\kappa_m}-\overline{\boldsymbol{\eta}}\cdot\right\}\frac{\nabla_t E_{pn}+p\kappa_m\mathbf{E}_{pt}}{u^{p+1}}\right]$$
$$-\left(2\kappa_m-\frac{\kappa_g}{\kappa_m}-\overline{\boldsymbol{\eta}}\cdot\right)\nabla_t E_n \qquad (12)$$

If we take a closer look at the term in the square brackets on the RHS of (12), we find that it can be written as

$$\frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}p\kappa_m\frac{\nabla_t E_{pn}+p\kappa_m\mathbf{E}_{pt}}{u^{p+1}}$$
$$+\left(jk_o+3\kappa_m-\frac{\kappa_g}{\kappa_m}-\overline{\boldsymbol{\eta}}\cdot\right)\left\{\hat{\mathbf{n}}\times\nabla\times\mathbf{E}-(jk_o+\kappa_m-\overline{\boldsymbol{\eta}}\cdot)\,\mathbf{E}_t\right\}$$

where we have substituted

$$\frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}\left[\frac{\nabla_t E_{pn}+p\kappa_m\mathbf{E}_{pt}}{u^{p+1}}\right]=\hat{\mathbf{n}}\times\nabla\times\mathbf{E}-(jk_o+\kappa_m-\overline{\boldsymbol{\eta}}\cdot)\,\mathbf{E}_t \qquad (13)$$

using the relation derived in (9).
Now the dominant terms on the RHS of (12) can be eliminated by considering the higher-order operator

$$\left[\hat{\mathbf{n}}\times\nabla\times-\left(jk_o+3\kappa_m-\frac{\kappa_g}{\kappa_m}-\overline{\boldsymbol{\eta}}\cdot\right)\right]\left\{\hat{\mathbf{n}}\times\nabla\times\mathbf{E}-(jk_o+\kappa_m-\overline{\boldsymbol{\eta}}\cdot)\,\mathbf{E}_t\right\}+$$
$$\left(2\kappa_m-\frac{\kappa_g}{\kappa_m}-\overline{\boldsymbol{\eta}}\cdot\right)\nabla_t E_n=\frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}p\kappa_m\,\frac{\nabla_t E_{pn}+p\kappa_m\mathbf{E}_{pt}}{u^{p+1}} \qquad (14)$$

The residual of (14) can be reduced further to yield the absorbing boundary condition of second order which satisfies (7) to $O(n^{-5})$. This second order

ABC is found to be

$$\left[\hat{\mathbf{n}} \times \nabla\times - \left(jk_o + 4\kappa_m - \frac{\kappa_g}{\kappa_m} - \overline{\boldsymbol{\eta}}\cdot\right)\right] \{\hat{\mathbf{n}} \times \nabla\times\mathbf{E} - (jk_o + \kappa_m - \overline{\boldsymbol{\eta}}\cdot)\mathbf{E}_t\} + \left(2\kappa_m - \frac{\kappa_g}{\kappa_m} - \overline{\boldsymbol{\eta}}\cdot\right)\nabla_t E_n = 0 \qquad (15)$$

and the residual is equal to

$$\frac{e^{-jk_o n}}{4\pi}\sum_{p=2}^{\infty}(p-1)\kappa_m \;\frac{\nabla_t E_{pn} + p\kappa_m \mathbf{E}_{pt}}{u^{p+1}} \qquad (16)$$

The operator on the LHS of (15) can be applied repeatedly to obtain ABCs of increasing order; however, higher order basis functions are needed for their implementation.

After some algebraic manipulation, the terms on the LHS of (15) reduced to simpler ones. In addition to the wave equation, the following vector identities were derived to carry out the simplifications and are provided below for the reader's convenience:

$$\begin{aligned} \hat{\mathbf{n}} \times \nabla\times\mathbf{E}_t &= \hat{\mathbf{n}} \times \nabla\times\mathbf{E} - \nabla_t E_n \\ \hat{\mathbf{n}} \times \nabla\times\nabla_t E_n &= \nabla_t(\nabla\cdot\mathbf{E}_t) + 2\kappa_m \nabla_t E_n \\ \hat{\mathbf{n}} \times \nabla\times(\hat{\mathbf{n}} \times \nabla\times\mathbf{E}) &= \nabla\times\{\hat{\mathbf{n}}(\nabla\times\mathbf{E})_n\} - k_o^2\mathbf{E}_t + \Delta\kappa\,\hat{\mathbf{n}} \times \nabla\times\mathbf{E} \end{aligned}$$

where $\Delta\kappa = \kappa_1 - \kappa_2$. The derivation of these identities is given in Appendix B. Upon simplification, the second order ABC can be compactly written as

$$-(D - \Delta\kappa - 2\overline{\boldsymbol{\eta}}\cdot)\,\hat{\mathbf{n}} \times \nabla\times\mathbf{E} + \left\{4\kappa_m^2 - \kappa_g + D(jk_o - \overline{\boldsymbol{\eta}}\cdot) + (\overline{\boldsymbol{\eta}})^2\cdot\right\}\mathbf{E}_t + \nabla\times\{\hat{\mathbf{n}}(\nabla\times\mathbf{E})_n\} + \left(jk_o + 3\kappa_m - \frac{\kappa_g}{\kappa_m} - 2\overline{\boldsymbol{\eta}}\cdot\right)\nabla_t E_n = 0 \quad (17)$$

in which

$$D = 2jk_o + 5\kappa_m - \frac{\kappa_g}{\kappa_m}$$

and

$$(\overline{\boldsymbol{\eta}})^2 \cdot \mathbf{E}_t = \kappa_1^2 E_{t_1}\hat{\mathbf{t}}_1 + \kappa_2^2 E_{t_2}\hat{\mathbf{t}}_2$$

The derivatives of the curvatures in (17) have been ignored due to the reasons outlined in [4]. These derivatives essentially give rise to second-order curvature terms which add to the coefficient of $\mathbf{E}_t$ only. The second order ABC derived in [1] is recovered on setting $\kappa_1 = \kappa_2 = 1/r$.

2.2 Symmetric correction

It has been shown by Peterson in [1] that the LHS of (17) when incorporated into the finite element equations gives rise to an unsymmetric matrix system in spherical coordinates. To alleviate this problem, Kanellopoulos and Webb[2] suggested an alternative derivation involving an arbitrary parameter which would lead to a symmetric matrix while sacrificing some accuracy. Below, we discuss a different approach which leads to a symmetric ABC without the introduction of an arbitrary parameter.

On considering the series expansion of the term $\hat{\mathbf{n}} \times \nabla\times\nabla_t E_n$, we have

$$\begin{aligned} \hat{\mathbf{n}} \times \nabla\times\nabla_t E_n &= \frac{e^{-jk_o n}}{4\pi}\sum_{p=1}^{\infty}\{jk_o + (p+1)\kappa_m\}\frac{\nabla_t E_{pn}}{u^{p+1}} \\ &= jk_o\nabla_t E_n + 2\kappa_m\nabla_t E_n + \sum_{p=2}^{\infty}(p-1)\kappa_m\frac{\nabla_t E_{pn}}{u^{p+1}} \\ &= jk_o\nabla_t E_n + 2\kappa_m\nabla_t E_n + O\left(n^{-5}\right) \end{aligned}$$

and on making use of the vector identity

$$\nabla_t(\nabla\cdot\mathbf{E}_t) = \hat{\mathbf{n}} \times \nabla\times\nabla_t E_n - 2\kappa_m\nabla_t E_n$$

given earlier, we arrive at the following result

$$\nabla_t(\nabla\cdot\mathbf{E}_t) = jk_o\nabla_t E_n + O\left(n^{-5}\right) \tag{18}$$

Since our ABC was derived to have a residual error of $O\left(n^{-5}\right)$, we can replace $jk_o\nabla_t E_n$ with $\nabla_t(\nabla\cdot\mathbf{E}_t)$ without affecting the order of the approximation. Doing so, the second order ABC with a symmetric operator can be rewritten as

$$\begin{aligned} (D - \Delta\kappa - 2\overline{\boldsymbol{\eta}}\cdot)\,\hat{\mathbf{n}} \times \nabla\times\mathbf{E} &= \left\{4\kappa_m^2 - \kappa_g + D\,(jk_o - \overline{\boldsymbol{\eta}}\cdot) + (\overline{\boldsymbol{\eta}})^2\cdot\right\}\mathbf{E}_t + \\ &\nabla\times\left\{\hat{\mathbf{n}}\,(\nabla\times\mathbf{E})_n\right\} + \frac{1}{jk_o}\left(jk_o + 3\kappa_m - \frac{\kappa_g}{\kappa_m} - 2\overline{\boldsymbol{\eta}}\cdot\right)\nabla_t(\nabla\cdot\mathbf{E}_t) \end{aligned} \tag{19}$$

It can be easily shown that the above boundary condition leads to a symmetric system of equations when incorporated into the finite element functional for surfaces having $\kappa_1 = \kappa_2$. Equations (10) and (19) reduce to the boundary conditions derived in [2] on setting $\kappa_1 = \kappa_2 = 1/r$ which have been found to work well for spherical and flat boundaries[7].

2.3 Finite element implementation

The boundary condition outlined in equation (19) cannot be incorporated into the finite element equations without modification. As explained in Chapter 3, the absorbing boundary condition is implemented in the finite element system through the surface integral over the mesh termination surface S_o.

$$\int_{S_o} \mathbf{E} \cdot \hat{\mathbf{n}} \times \nabla\times\mathbf{E} \, dS \;=\; \int_{S_o} \mathbf{E} \cdot P(\mathbf{E}) \, dS$$

where $P(\mathbf{E})$ denotes the boundary condition relating the tangential magnetic field to the tangential electric field on the surface.

Let $P_1(\mathbf{E})$ denote the first order absorbing boundary condition given by (10), where the subscript represents the order of the ABC. Therefore, the surface integral contribution for the first order ABC reduces to

$$\int_{S_o} \mathbf{E} \cdot P_1(\mathbf{E}) \, dS \;=\; (jk_o + \kappa_m) \int_{S_o} \mathbf{E} \cdot \mathbf{E}_t \, dS - \int_{S_o} \mathbf{E} \cdot (\overline{\eta} \cdot \mathbf{E}_t) \, dS \quad (20)$$

Using some basic vector identities and considering that $\mathbf{E}_t = -\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}$, we deduce that

$$\int_{S_o} \mathbf{E} \cdot P_1(\mathbf{E}) \, dS \;=\; (jk_o + \kappa_m) \int_{S_o} \left(E_{t_1}^2 + E_{t_2}^2\right) dS - \int_{S_o} \left(\kappa_1 E_{t_1}^2 + \kappa_2 E_{t_2}^2\right) dS \quad (21)$$

which is a readily implementable form of the first order ABC. However, the second order ABC does not simplify as easily. If $P_2(\mathbf{E})$ denotes the second order ABC given by (19), we can rewrite it in more compact vector notation as

$$P_2(\mathbf{E}) \;=\; \overline{\alpha} \cdot \mathbf{E}_t + \overline{\beta} \cdot [\nabla\times \{\hat{\mathbf{n}} (\nabla\times\mathbf{E})_n\}] + \overline{\gamma} \cdot \{\nabla_t (\nabla\cdot\mathbf{E}_t)\} \quad (22)$$

where the tensors $\overline{\alpha}, \overline{\beta}$ and $\overline{\gamma}$ are given by

$$\begin{aligned}\overline{\alpha} \;&=\; \frac{1}{D - \Delta\kappa - 2\kappa_1} \left\{4\kappa_m^2 - \kappa_g + D\,(jk_o - \kappa_1) + \kappa_1^2\right\} \hat{\mathbf{t}}_1\hat{\mathbf{t}}_1 \\ &\quad + \frac{1}{D - \Delta\kappa - 2\kappa_2} \left\{4\kappa_m^2 - \kappa_g + D\,(jk_o - \kappa_2) + \kappa_2^2\right\} \hat{\mathbf{t}}_2\hat{\mathbf{t}}_2 \quad (23)\end{aligned}$$

$$\overline{\beta} \;=\; \frac{1}{D - \Delta\kappa - 2\kappa_1} \hat{\mathbf{t}}_1\hat{\mathbf{t}}_1 + \frac{1}{D - \Delta\kappa - 2\kappa_2} \hat{\mathbf{t}}_2\hat{\mathbf{t}}_2 \quad (24)$$

$$
\begin{aligned}
\overline{\gamma} &= \frac{1}{jk_o(D-\Delta\kappa-2\kappa_1)}\left(jk_o+3\kappa_m-\frac{\kappa_g}{\kappa_m}-2\kappa_1\right)\hat{\mathbf{t}}_1\hat{\mathbf{t}}_1 \\
&+\frac{1}{jk_o(D-\Delta\kappa-2\kappa_2)}\left(jk_o+3\kappa_m-\frac{\kappa_g}{\kappa_m}-2\kappa_2\right)\hat{\mathbf{t}}_2\hat{\mathbf{t}}_2 \qquad (25)
\end{aligned}
$$

Substituting the second-order absorbing boundary condition in the surface integral given in (22), we have

$$
\begin{aligned}
\int_{S_o}\mathbf{E}\cdot P_2(\mathbf{E})\,dS &= \int_{S_o}\mathbf{E}\cdot(\overline{\alpha}\cdot\mathbf{E}_t)\;dS+\int_{S_o}\mathbf{E}\cdot\left(\overline{\beta}\cdot\{\hat{\mathbf{n}}\,(\nabla\times\mathbf{E})_n\}\right)\;dS \\
&+\int_{S_o}\mathbf{E}\cdot\{\overline{\gamma}\cdot\nabla_t\,(\nabla\cdot\mathbf{E}_t)\}\;dS \\
&= I_1+I_2+I_3
\end{aligned}
$$

Let us examine the integral I_1. Since $\mathbf{E}_t=-\hat{\mathbf{n}}\times\hat{\mathbf{n}}\times\mathbf{E}$, we have

$$
I_1 = \int_{S_o}\alpha_1E_{t_1}^2+\alpha_2E_{t_2}^2\;dS \qquad (26)
$$

after employing some simple vector identities.

The other two integrals (I_2 and I_3) do not reduce as easily to simple, implementable forms. They are first simplified using basic vector and tensor identities and then the divergence theorem is employed to eliminate one of the terms. Considering the integrand of the second integral I_2, we note that

$$
\mathbf{E}\cdot\left[\overline{\beta}\cdot(\nabla\times\hat{\mathbf{n}}\phi)\right] = \left(\overline{\beta}\cdot\mathbf{E}\right)\cdot(\nabla\times\hat{\mathbf{n}}\phi)
$$

where we have set $\phi=(\nabla\times\mathbf{E})_n$. Using some additional vector identities and letting $\overline{\beta}\cdot\mathbf{E}=\boldsymbol{F}$, we get

$$
\begin{aligned}
\boldsymbol{F}\cdot\nabla\times\hat{\mathbf{n}}\phi &= \nabla\cdot(\phi\hat{\mathbf{n}}\times\boldsymbol{F})+\phi\hat{\mathbf{n}}\cdot\nabla\times\boldsymbol{F} \\
&= \nabla\cdot(\phi\hat{\mathbf{n}}\times\boldsymbol{F})+\phi\,(\nabla\times\boldsymbol{F})_n
\end{aligned}
$$

Using the results from [3], the first term in the above identity can be further simplified to read

$$
\begin{aligned}
\nabla\cdot(\phi\hat{\mathbf{n}}\times\boldsymbol{F}) &= \nabla_s\cdot(\phi\hat{\mathbf{n}}\times\boldsymbol{F})+\frac{\partial}{\partial n}\{\phi\hat{\mathbf{n}}\cdot(\hat{\mathbf{n}}\times\boldsymbol{F})\}-J\{\phi\hat{\mathbf{n}}\cdot(\hat{\mathbf{n}}\times\boldsymbol{F})\} \\
&= \nabla_s\cdot(\phi\hat{\mathbf{n}}\times\boldsymbol{F}) \qquad (27)
\end{aligned}
$$

where ∇_s denotes the surface gradient operator and $J = \kappa_1 + \kappa_2$. The integral I_2 can now be written as

$$I_2 = \int_{S_o} \nabla_s \cdot (\phi \hat{\mathrm{n}} \times \boldsymbol{F}) \, dS + \int_{S_o} \phi (\nabla \times \boldsymbol{F})_n \, dS$$

We can now apply the surface divergence theorem to the first term on the RHS of the this expression to yield

$$\int_{S_o} \nabla_s \cdot (\phi \hat{\mathrm{n}} \times \boldsymbol{F}) \, dS = \int_C \phi \hat{\mathrm{m}} \cdot (\hat{\mathrm{n}} \times \boldsymbol{F}) \, dl = 0 \tag{28}$$

since the surface S_o is closed. We note that $\hat{\mathrm{m}} = \hat{\mathrm{l}} \times \hat{\mathrm{n}}$ and $\hat{\mathrm{l}}$ is the unit vector along the edge of the surface element and C denotes the contour of integration. On the basis of (28), I_2 reduces to

$$I_2 = \int_{S_o} (\nabla \times \mathbf{E})_n \left\{ \nabla \times (\overline{\beta} \cdot \mathbf{E}) \right\}_n \, dS \tag{29}$$

We now turn our attention to simplifying I_3 for implementation in the finite element equations. Considering the integrand of I_3, we have

$$\begin{aligned} \mathbf{E} \cdot \{ \overline{\gamma} \cdot \nabla_t (\nabla \cdot \mathbf{E}_t) \} &= (\overline{\gamma} \cdot \mathbf{E}) \cdot \{ \nabla_t (\nabla \cdot \mathbf{E}_t) \} \\ &= (\overline{\gamma} \cdot \mathbf{E}) \cdot \left\{ \nabla \psi - \hat{\mathrm{n}} \frac{\partial \psi}{\partial n} \right\} \end{aligned}$$

where $\psi = \nabla \cdot \mathbf{E}_t$. Next, setting $\boldsymbol{G} = \overline{\gamma} \cdot \mathbf{E}$, we obtain

$$\boldsymbol{G} \cdot \left\{ \nabla \psi - \hat{\mathrm{n}} \frac{\partial \psi}{\partial n} \right\} = \nabla \cdot (\psi \boldsymbol{G}) - \psi \nabla \cdot \boldsymbol{G} - G_n \frac{\partial \psi}{\partial n} \tag{30}$$

The first term in the above identity can be written as

$$\nabla \cdot (\psi \boldsymbol{G}) = \nabla_s \cdot (\psi \boldsymbol{G}) + \frac{\partial}{\partial n} (\psi G_n) - J (\psi G_n)$$

and since $\partial G_n / \partial n = \nabla \cdot \boldsymbol{G} - \nabla \cdot \boldsymbol{G}_t + J G_n$, the LHS of (30) reduces to

$$\boldsymbol{G} \cdot \left\{ \nabla \psi - \hat{\mathrm{n}} \frac{\partial \psi}{\partial n} \right\} = \nabla_s \cdot (\psi \boldsymbol{G}) - \psi \nabla \cdot \boldsymbol{G}_t \tag{31}$$

We now replace the integrand of I_3 with the expression in (31) and use the divergence theorem to eliminate the first term of (31). Specifically,

$$\begin{aligned} I_3 &= \int_{S_o} \nabla_s \cdot (\psi \boldsymbol{G})\, dS - \int_{S_o} \psi \nabla \cdot \boldsymbol{G}_t\, dS \\ &= \int_C \hat{\mathrm{m}} \cdot (\psi \boldsymbol{G})\, dS - \int_{S_o} \psi \nabla \cdot \boldsymbol{G}_t\, dS = -\int_{S_o} \psi \nabla \cdot \boldsymbol{G}_t\, dS \end{aligned}$$

where $\hat{\mathrm{m}}$ has been defined earlier and the contour integral vanishes since the surface is closed. The integral I_3 can finally be rewritten as

$$I_3 \;=\; -\int_{S_o} (\nabla \cdot \mathbf{E}_t)(\nabla \cdot \boldsymbol{G}_t)\; dS \qquad (32)$$

Using (26), (29) and (32), the complete surface integral term incorporating the conformal second order ABC reduces to

$$\begin{aligned} \int_{S_o} \mathbf{E} \cdot P_2(\mathbf{E})\, dS \;=\; & \int_{S_o} \left(\alpha_1 E_{t_1}^2 + \alpha_2 E_{t_2}^2\right)\, dS + \int_{S_o} (\nabla \times \mathbf{E})_n \left\{\nabla \times (\overline{\boldsymbol{\beta}} \cdot \mathbf{E})\right\}_n\, dS \\ & - \int_{S_o} (\nabla \cdot \mathbf{E}_t)\left\{\nabla \cdot (\overline{\gamma} \cdot \mathbf{E})_t\right\}\, dS \end{aligned} \qquad (33)$$

It remains to be seen whether the integrals in (33) lead to a symmetric system when incorporated into the finite element equations. With this in mind, we will examine three simple shapes and check whether they preserve symmetry of the finite element system. It will then be possible to generalize our findings to a more general mesh truncation boundary.

Let us consider the case of a sphere of radius r. Since the two principal curvatures of the sphere are identical ($\kappa_1 = \kappa_2 = 1/r$), the first order boundary condition reduces to the simple Sommerfeld radiation condition

$$\int_{S_o} \mathbf{E} \cdot P_1(\mathbf{E})\, dS \;=\; jk_o \int_{S_o} \left(E_\theta^2 + E_\phi^2\right)\, dS \qquad (34)$$

On a spherical boundary, the second order ABC also reduces to the comparatively simple form:

$$\int_{S_o} \mathbf{E} \cdot P_2(\mathbf{E})\, dS \;=\; \int_{S_o} \left[jk_o \mathbf{E}_t^2 + \frac{1}{2jk_o + 2/r} (\nabla \times \mathbf{E})_n^2 - \frac{1}{2jk_o + 2/r} (\nabla \cdot \mathbf{E}_t)^2 \right] dS \qquad (35)$$

The ABC given in (35) is identical to the boundary condition derived in [2] for a spherical mesh termination surface and leads to a symmetric system of equations.

Next, we consider the case of a planar termination boundary in which case $\kappa_1 = \kappa_2 = 0$. The first order ABC then reduces to the Sommerfeld radiation condition and the second order ABC for a planar boundary simplifies to

$$\int_{S_o} \mathbf{E} \cdot P_2(\mathbf{E})\, dS = \int_{S_o} \left[jk_o \mathbf{E}_t^2 + \frac{1}{2jk_o} (\nabla \times \mathbf{E})_n^2 - \frac{1}{2jk_o} (\nabla \cdot \mathbf{E}_t)^2 \right] dS \quad (36)$$

Since the planar boundary is a special case of a spherical boundary, (36) again reduces to asymmetric system of equations.

Now we examine the situation when the mesh termination boundary is cylindrical in shape and of radius ρ. The principal curvatures of the cylindrical surface are then $\kappa_1 = 1/\rho$ and $\kappa_2 = 0$. Since the principal curvatures are no longer identical, the tensors $\overline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\beta}}$ and $\overline{\boldsymbol{\gamma}}$ do not reduce to simple scalars. The first order ABC for a cylindrical outer boundary is given by

$$\int_{S_o} \mathbf{E} \cdot P_1(\mathbf{E})\, dS = \left(jk_o + \frac{1}{\rho}\right) \int_{S_o} \left(jk_o + \frac{1}{\rho}\right) \mathbf{E}_t^2\, dS - \int_{S_o} \frac{1}{\rho} E_\phi^2\, dS \quad (37)$$

and the second order ABC gives by

$$\int_{S_o} \mathbf{E} \cdot P_2(\mathbf{E})\, dS = \int_{S_o} \left(\alpha_{c1} E_\phi^2 + \alpha_{c2} E_z^2\right) dS + \int_{S_o} (\nabla \times \mathbf{E})_\rho \left\{\nabla \times (\overline{\boldsymbol{\beta}}_c \cdot \mathbf{E})\right\}_\rho dS$$
$$- \int_{S_o} (\nabla \cdot \mathbf{E}_t) \{\nabla \cdot (\overline{\boldsymbol{\gamma}}_c \cdot \mathbf{E})_t\}\, dS \quad (38)$$

where $\alpha_{c1}, \alpha_{c2}, \overline{\boldsymbol{\beta}}_c$ and $\overline{\boldsymbol{\gamma}}_c$ are obtained by substituting $\kappa_1 = 1/\rho$ and $\kappa_2 = 0$ in the original expressions for $\overline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\beta}}$ and $\overline{\boldsymbol{\gamma}}$. It is seen that the first order ABC given by (37) leads to a symmetric matrix for a cylindrical boundary. On the other hand, the second order ABC does not yield a symmetric matrix for an arbitrary choice of basis functions. However, the boundary condition outlined in (38) preserves symmetry on using linear edge-based elements for discretization.

The above discussion enables us to conclude that the first order boundary condition leads to a symmetric system for surfaces having arbitrary principal curvatures. However, symmetry is guaranteed for the second order ABC only when the two principal curvatures of the mesh termination boundary are identical, i.e., only when the outer boundary is limited to a planar or a spherical surface. Thus if we want to enclose a scatterer having arbitrary shape within a conformal outer boundary, an unsymmetric system of equations will have to be solved. It should, however, be noted that the resulting unsymmetric system will, in general, have a lesser number of unknowns than its symmetric counterpart.

3 Applications

In the previous section, we have discussed the derivation of absorbing boundary conditions which can be employed on surfaces conformal to the scattering or radiating structure. As a result, the mesh termination boundary can be made to enclose the scattering object more snugly. Consequently for arbitrary targets, we achieve a substantial saving in the amount of volume to be meshed between the ABC surface and that of the scatterer. This is particularly critical when the target is cylindrical in shape or a combination of cylindrical, doubly curved and planar surfaces as is the case with any real-life structure.

In this section, we examine the performance of these boundary conditions when applied on conformal mesh termination surfaces.

A. Composite cube

For our first example, we compute the backscatter pattern of the half metal-half dielectric cube geometry shown in Chapter 3. However, instead of using a spherical surface to terminate the mesh, we employ the absorbing boundary condition on a piecewise planar surface, i.e., a cubical box placed only 0.3λ from the face of the scatterer. The geometry is shown in Figure 2 and needed only 30,000 unknowns for discretization. This is in stark contrast to the 40,000 unknown system which resulted when the geometry was enclosed in a spherical termination boundary. The decrease in the unknown count is even more dramatic as we go to larger scatterers. In Figure 3, we plot the backscatter pattern in the $x-z$ plane ($E_z^{inc} = 0$ polarization) for the metal-dielectric cube geometry given in Figure 2 and compare the computed values with data obtained from a traditional method of moments (MoM) code[8]. The dielectric-filled section has unit permeability and a relative permittivity of $2 - j2$. The agreement with reference data is seen to be excellent; it can therefore be concluded that accuracy of the far-field values

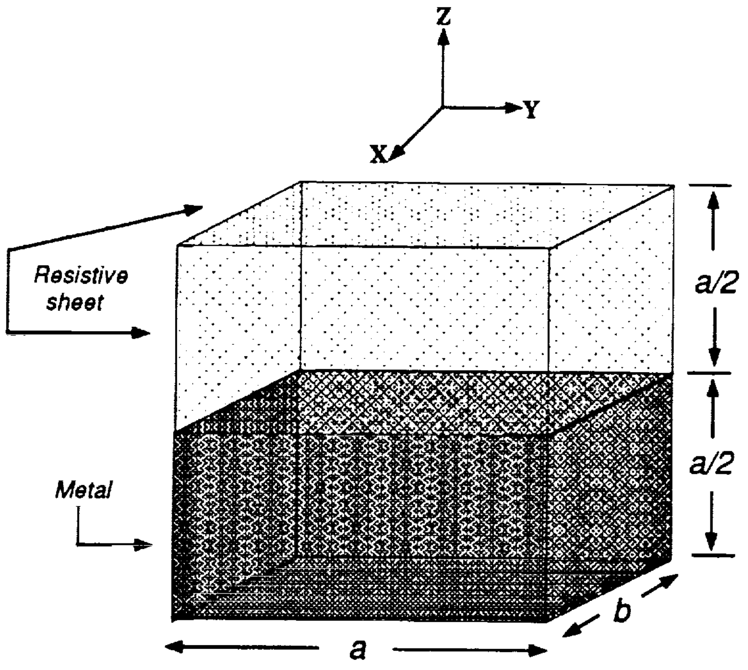


Figure 2: Geometry of cube ($a = b = 0.5\lambda$) consisting of a metallic section and a dielectric section ($\epsilon_r = 2 - j2$), where the latter is bounded by a resistive surface having $R = Z_o$.

has not been affected by a different mesh termination scheme. In fact, we have obtained results of comparable accuracy with only 75% of the computing resources than were necessary before. This observation will be made by the reader again and again in the following pages as the full capability of these conformal ABCs is demonstrated.

B. Inlets

In our next example, we compute the scattering from perfectly conducting inlets. The aperture of an inlet usually has a large radar cross-section around normal incidence: therefore, a good understanding of its scattering characteristics is critical if measures need to be taken for reducing its echo-area. An accurate computer simulation of such a geometry provides a cost-effective and ready way of allowing the designer to experiment with complex material fillings to achieve satisfactory results. All our validations are carried out for empty inlets due to lack of reference data for more complicated structures.

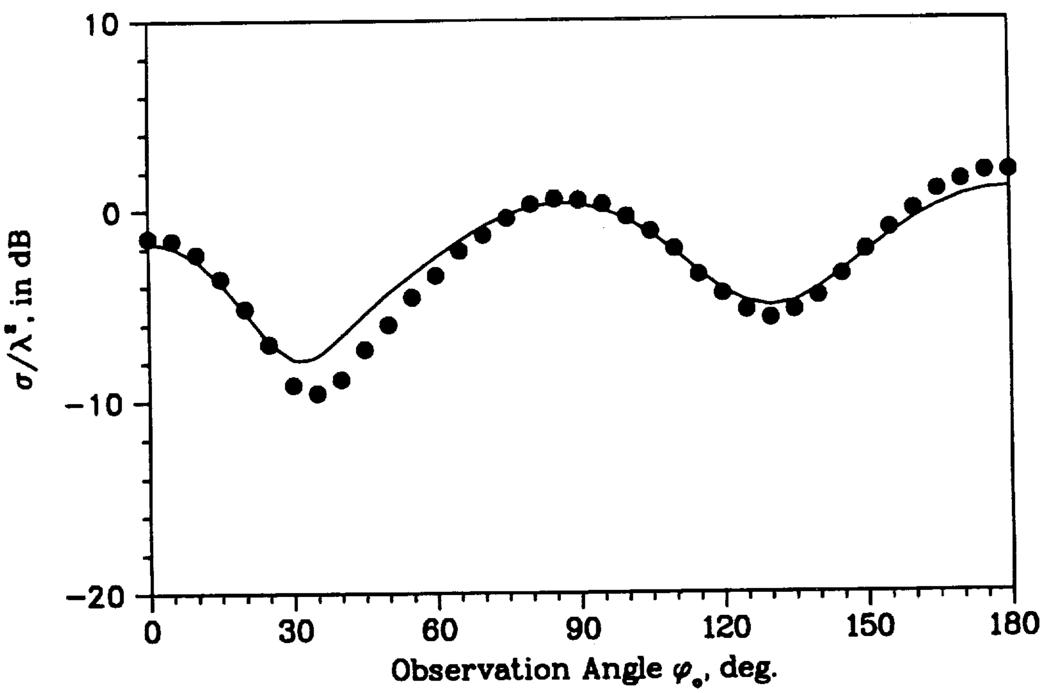


Figure 3: RCS pattern in the $x - z$ plane for the composite cube shown earlier. The solid curve is the FEMATS pattern and the black dots are MoM data for the $E_z^{inc} = 0$ polarization. Mesh termination is piecewise planar.

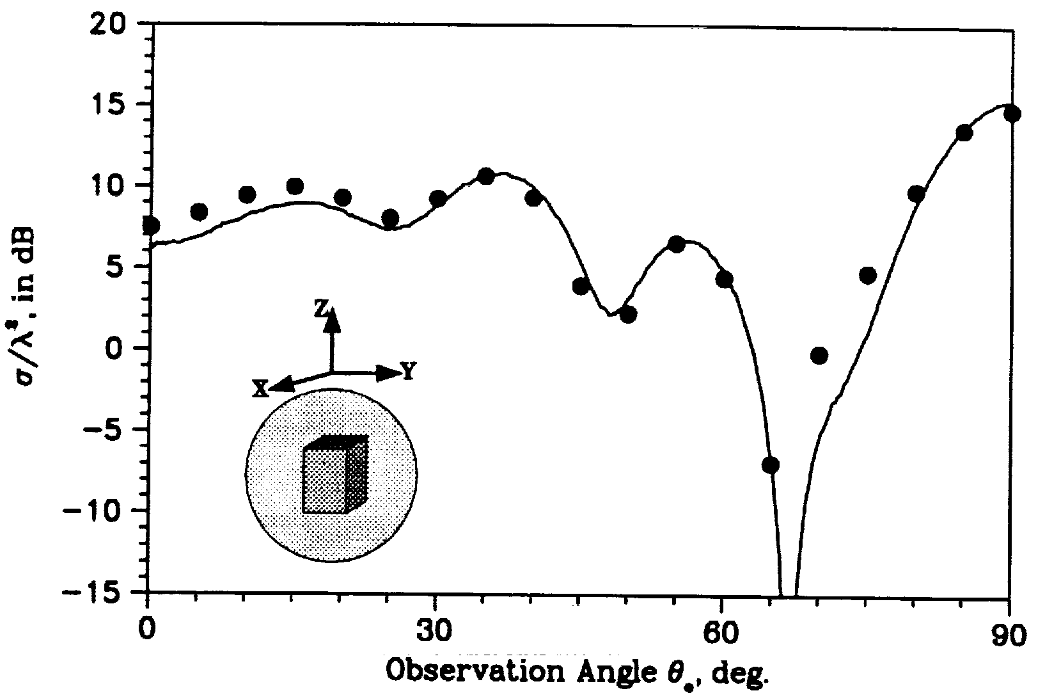


Figure 4: Backscatter pattern of a metallic rectangular inlet $(1\lambda \times 1\lambda \times 1.5\lambda)$ for HH polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is spherical.

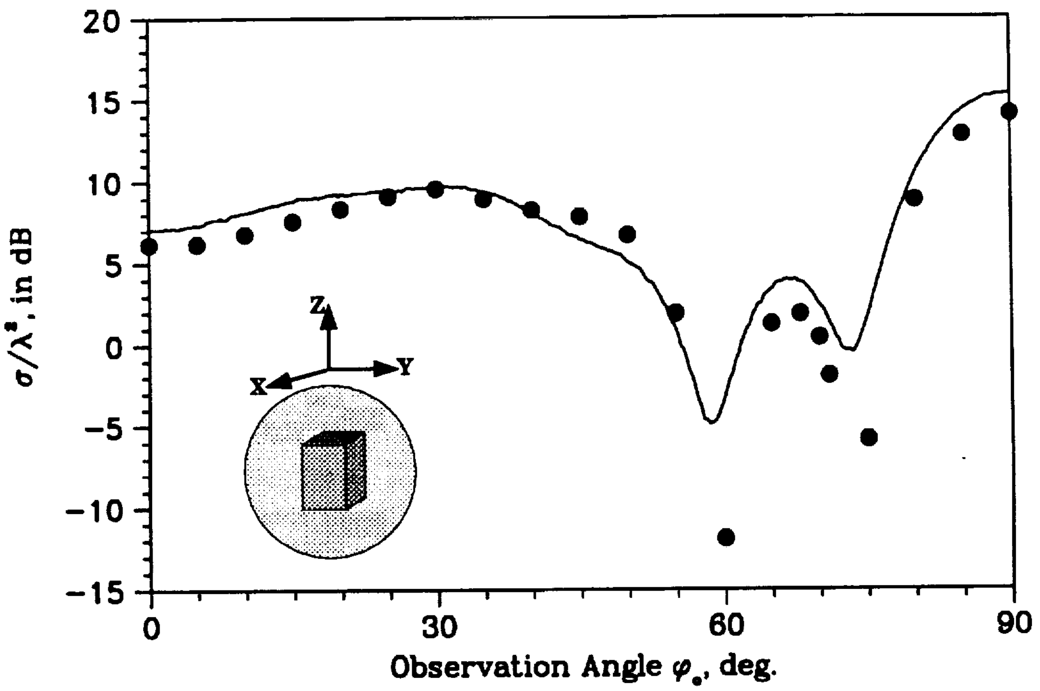


Figure 5: Backscatter pattern of a metallic rectangular inlet ($1\lambda \times 1\lambda \times 1.5\lambda$) for VV polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is spherical.

The geometry of interest is a perfectly conducting rectangular inlet, with dimensions $1\lambda \times 1\lambda \times 1.5\lambda$. For the plots shown in Figures 4 and 5, we have enclosed the target within a sphere of radius 1.35λ, which is only about $.35\lambda$ from the farthest edge of the scatterer. This resulted in a system of 224,476 unknowns and converged in an average of 785 seconds per incidence angle on the 56 processor KSR1. The computed values from our code agrees very well with measured data for both HH and VV polarizations. As can be seen from the above discussion, we have obtained our solution using significant computing resources and time.

Our next step is to use the conformal mesh termination scheme formulated in the previous section and utilized in Example A. Therefore, instead of using a spherical mesh truncation surface, we terminate the mesh with a rectangular box placed only 0.35λ away from the scatterer (see inset of Figure 6). The problem size reduces dramatically to 145,000 unknowns, a 35% reduction over the spherical mesh termination scheme. The convergence time for each excitation vector is about 220 seconds, less than 4 minutes, when run on all 56 processors of the KSR1. The computed values are again compared with measured data for both polarizations in Figures 6 and 7; the agreement is excellent, albeit a bit worse than the spherical case. However, this fact is overshadowed by the fact that we have reduced the problem size by more than a third and computing time by about a fourth.

We then considered the problem of scattering from a perfectly conducting cylindrical inlet. Even though integral equation codes are more efficient for such bodies of revolution, our primary concern in this test was to examine the performance of the conformal absorbing boundary conditions that we derived earlier. The target is a perfectly conducting cylindrical inlet having a diameter of 1.25λ and a height of 1.875λ. We first used a rectangular outer boundary, placed $.45\lambda$ from the farthest edge of the target, to enclose the scatterer. The radar cross-section was then computed for a ϕ-polarized incident wave in the yz-plane and compared with measured data. The agreement was found to be quite good for all lobes except the third. We expect the results to improve on moving the outer boundary farther away.

Next, we used a truly conformal termination scheme by using a cylindrical surface for mesh truncation. It should be noted that this is the first instance of a non-spherical surface (i.e., a surface having different principal curvatures) being applied to terminate a finite element mesh for solving open problems. The cylindrical outer boundary was placed about 0.45λ from the target and

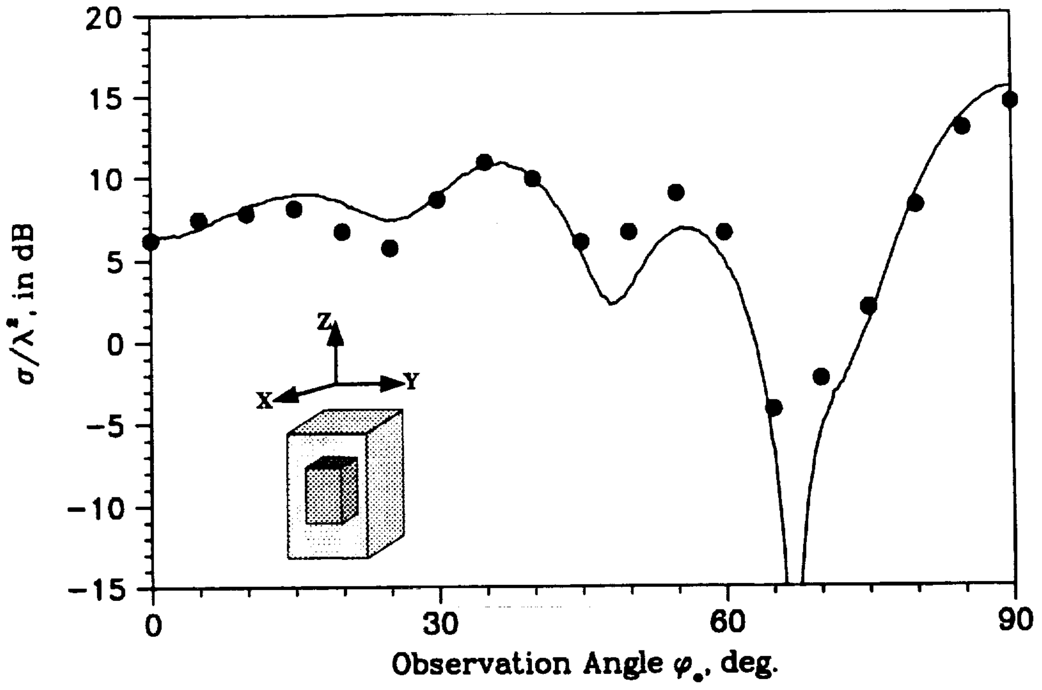


Figure 6: Backscatter pattern of a metallic rectangular inlet ($1\lambda \times 1\lambda \times 1.5\lambda$) for HH polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is piecewise planar.

RCS computations were carried out for a ϕ polarized incident wave and compared with measured data (Figure 8). The savings in computational cost for the cylindrical termination scheme as opposed to the rectangular mesh truncation boundary is quite impressive. The cylindrical mesh termination has only 144,392 unknowns compared to the 191,788 unknowns for a rectangular truncation scheme. A spherical mesh termination would have swelled to about 265,000 unknowns, sampling density and outer boundary distance remaining the same. Thus we have reduced the problem size by about 45% and computation time by a similar, if not greater, amount by using a conformal mesh termination scheme. The savings in computational resources is quite significant even when we compare the rectangular and cylindrical termination schemes - a 25% reduction in problem size and a similar decrease in computation time. In Figure 9, we plot the backscatter pattern for the same cylindrical inlet in which the incident wave is θ polarized. The agreement is

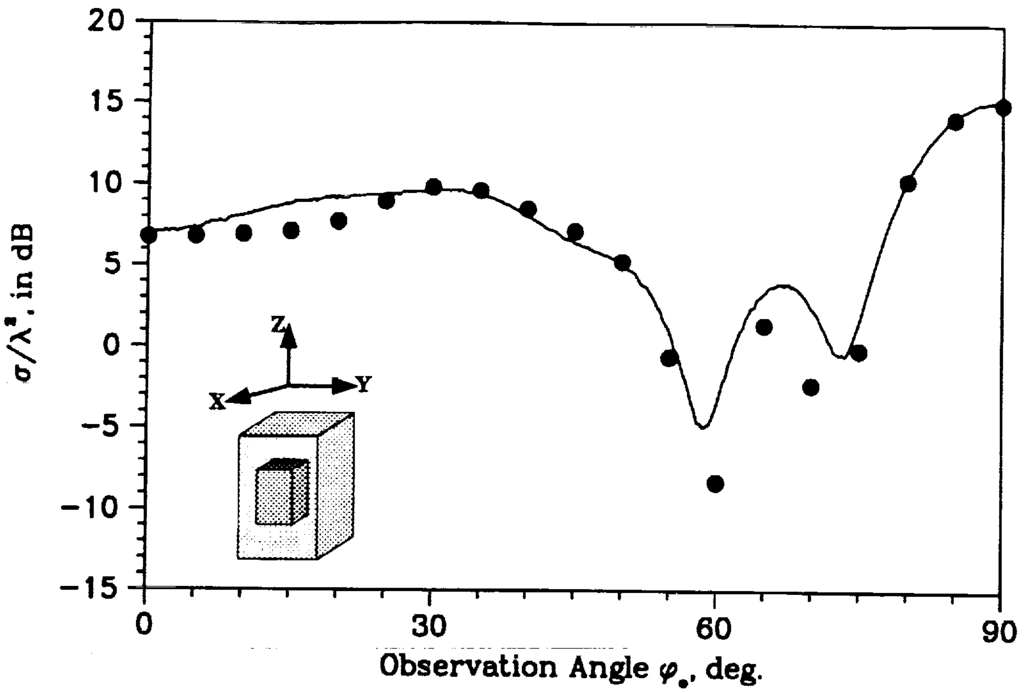


Figure 7: Backscatter pattern of a metallic rectangular inlet ($1\lambda \times 1\lambda \times 1.5\lambda$) for VV polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is piecewise planar.

seen to be decent for the entire range of incident angles.

C. Perfectly conducting plate

The motivation for testing the FEMATS code on the perfectly conducting plate was two-fold. It is usually very difficult to model the scattering from the edges of the plate even using integral equation methods. Therefore, we wanted to carry out some tests to see how the code would behave at edge-on incidence. Secondly, we wanted to examine the performance of termination boundaries of esoteric shapes. The first choice was to enclose the plate in a rectangular box. The second choice was to use a box with half cylinders attached to the faces normal to the plane of the plate - the reasoning being that since the edge of the plate behaves like a line source and scatters cylindrical waves, a cylindrical mesh termination was most suitable for wave absorption. It should be noted that both mesh termination schemes require approximately the same number of unknowns; the superiority of one over the

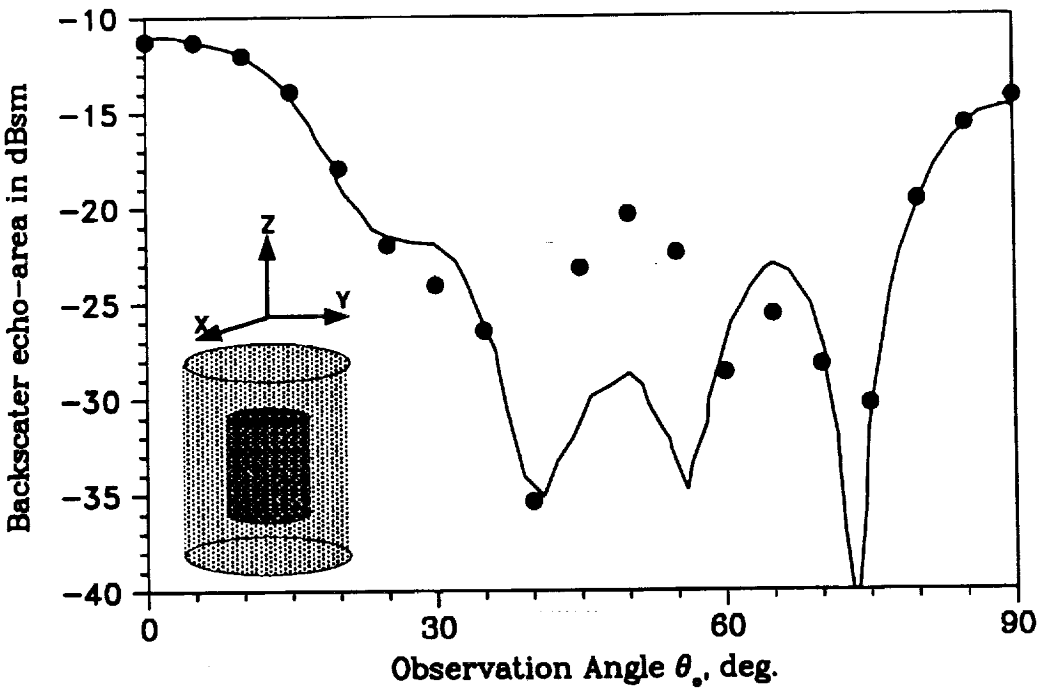


Figure 8: Backscatter pattern of a perfectly conducting cylindrical inlet (diameter=1.25λ, height=1.875λ) for HH polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is a circular cylinder.

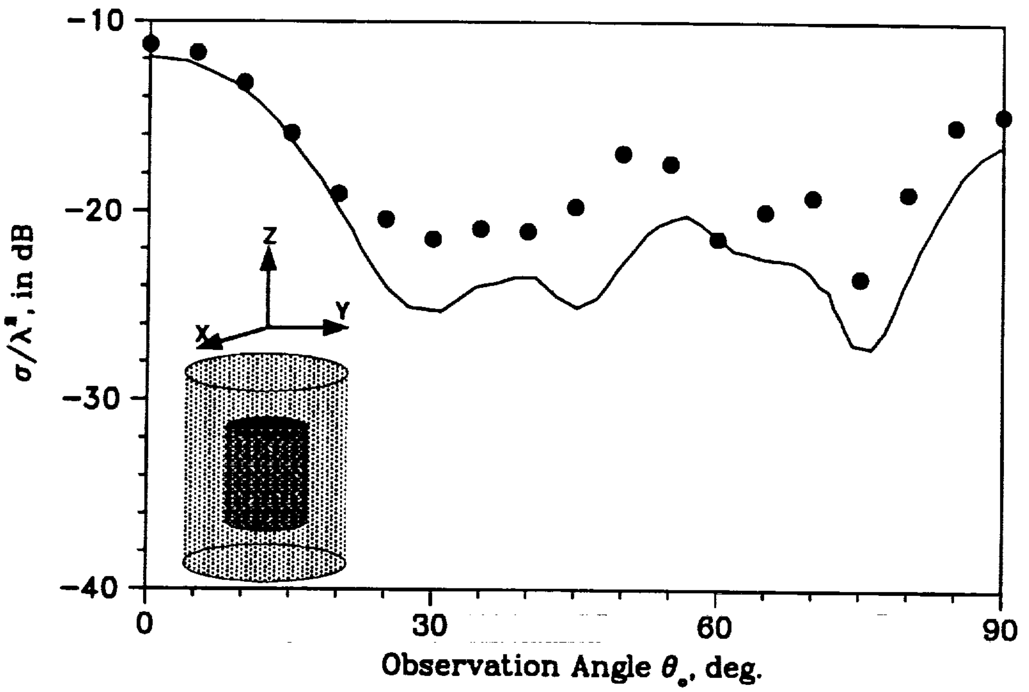


Figure 9: Backscatter pattern of a perfectly conducting cylindrical inlet (diameter=1.25λ, height=1.875λ) for HH polarization. Black dots indicate computed values and the solid line represents measured data. Mesh termination surface is a circular cylinder.

other will thus be decided only on the basis of computed backscatter values.

Our test case is a $3.5\lambda \times 2\lambda$ perfectly conducting rectangular plate. In Figure 10, we plot the backscatter pattern for the $\theta\theta$ polarization in the xz plane, i.e., over the long side of the plate. The computed values compare very

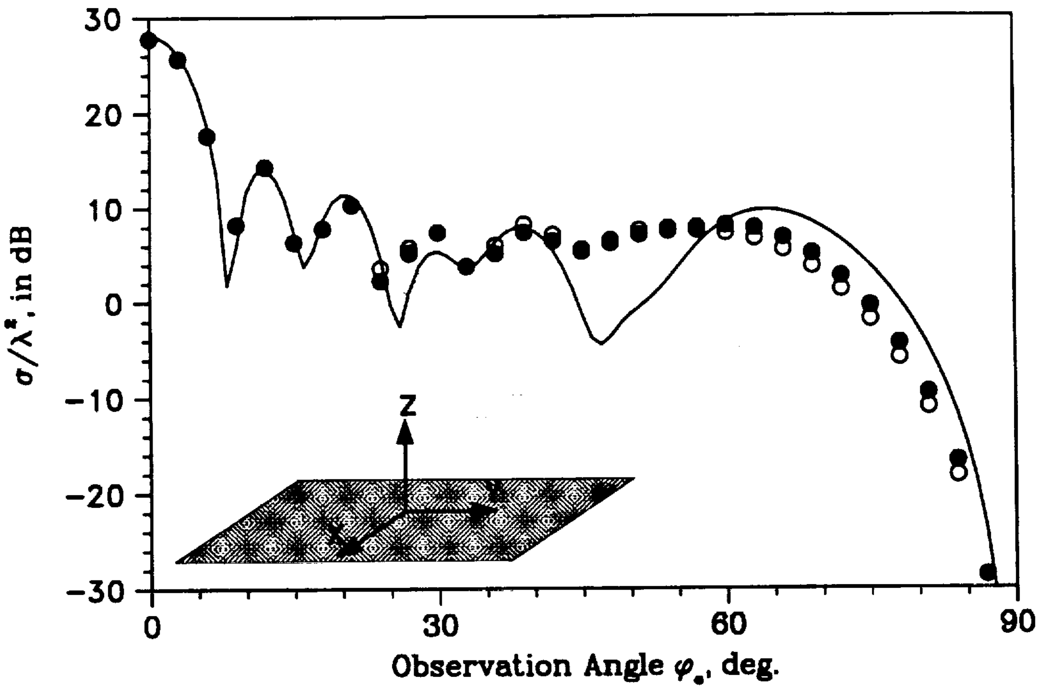


Figure 10: Backscatter pattern ($\sigma_{\theta\theta}$) of a $3.5\lambda \times 2\lambda$ perfectly conducting plate in the xz plane. The white dots indicate box termination; the black dots represent a combined box-cylinder termination.

well with reference data; however, the code does not pick up the sharp null at $\theta_o = 45^0$ and the two mesh termination schemes perform as well, although a slight improvement is noticeable for the box-cylinder termination.

Next, we plot the backscatter pattern of the same geometry for the $\phi\phi$ polarization over the long side of the plate in Figure 11. Again, the agreement with reference data is quite good. However, the backscatter echo-area at edge-on incidence is not calculated accurately.

In the next figure, we compute the RCS of the conducting plate in the yz plane, i.e., over its short side, for the $\phi\phi$ polarization. The backscatter echo-area for edge-on incidence is picked up very well for a rectangular-cylindrical

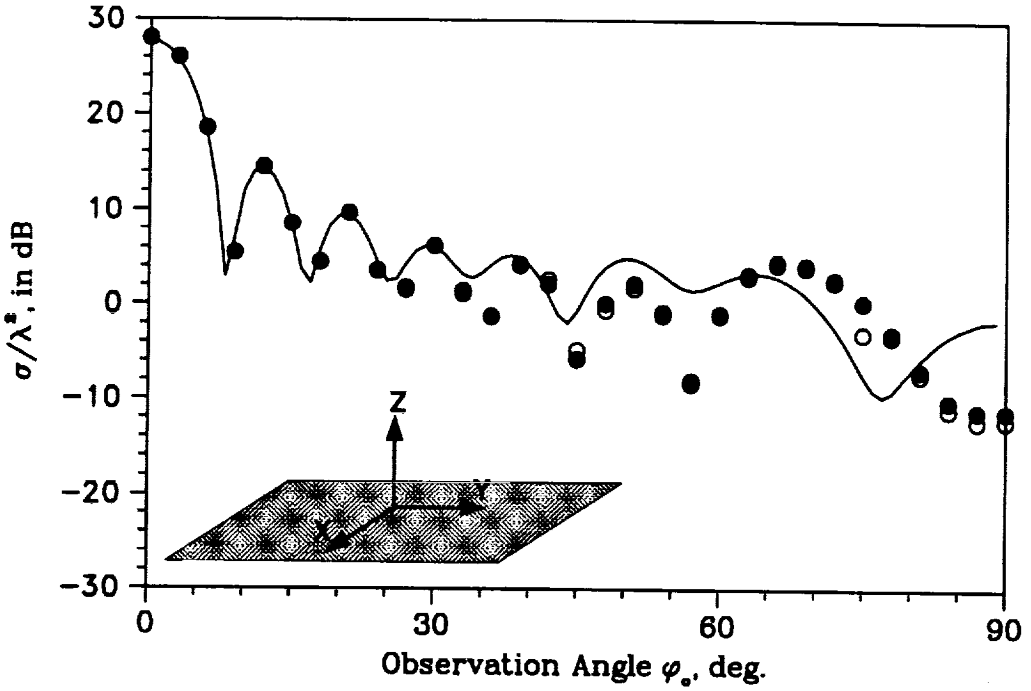


Figure 11: Backscatter pattern ($\sigma_{\phi\phi}$) of a $3.5\lambda \times 2\lambda$ perfectly conducting plate in the xz plane. The white dots indicate box termination; the black dots represent a combined box-cylinder termination.

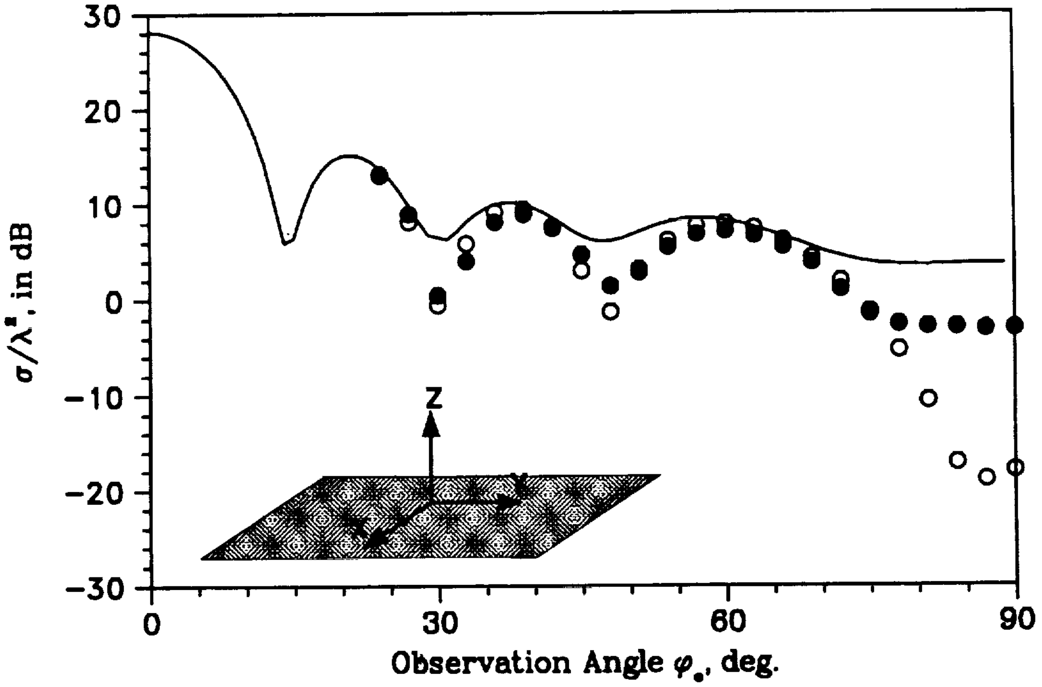


Figure 12: Backscatter pattern ($\sigma_{\phi\phi}$) of a 3.5λ × 2λ perfectly conducting plate in the yz plane. The white dots indicate box termination; the black dots represent a combined box-cylinder termination.

termination whereas a rectangular truncation scheme gives completely incorrect results. These two schemes have approximately the same storage requirement; in fact, the box-cylinder combination yields a slightly smaller system of equation. This example truly illustrates the power of a conformal truncation scheme composed of simple shapes: not only are the results far more accurate but even the storage requirement is slightly less.

In the last experiment, we compute the backscatter values of the conducting plate for a conical $\theta = 80^0$ cut (Figure 13). This problem is extremely difficult since the ABCs are usually the least accurate at edge-on. Even then, the comparison with reference data is decent for the box-cylinder combination and not so accurate for the rectangular truncation scheme. This is especially noticeable at $\phi = 0^0$ where the rectangular termination boundary yields completely inaccurate results.

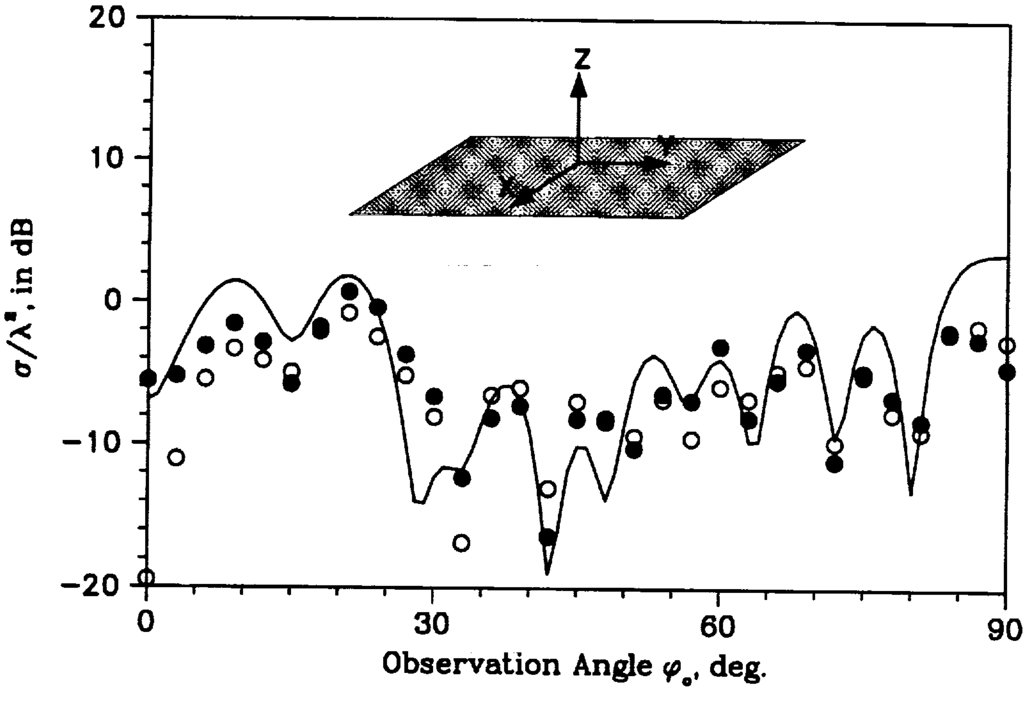


Figure 13: Backscatter pattern ($\sigma_{\phi\phi}$) of a $3.5\lambda \times 2\lambda$ perfectly conducting plate for a conical $\theta = 80^0$ cut. The white dots indicate box termination; the black dots represent a combined box-cylinder termination.

In all the above simulations, the boundary was terminated at 0.35λ from the flat face of the plate and 0.5λ from the edges of the plate.

D. Glass plate

The next example presents the computation of radar cross-section of a $1\lambda \times 1.75\lambda \times .125\lambda$ glass plate having a relative permittivity of 5.889 -j 0.0551. The target was enclosed in a rectangular outer boundary placed only 0.3λ from the edge of the plate. This resulted in a system of about 146,000 unknowns which required an average of 8,000 iterations to converge per incidence angle for a diagonally preconditioned biconjugate gradient solver. The relatively slow convergence is characteristic of a high contrast, pure dielectric scatterer. It should be noted that a metal-backed dielectric target converges to the same tolerance in less than half the number of iterations required for a pure dielectric one. In Figure 14, we plot the backscatter pattern of the glass plate for the VV polarization on the z-x plane and compare it with results obtained from a method of moments code[9]. The agreement is decent considering that the method of moments code was run only with 4 samples per wavelength.

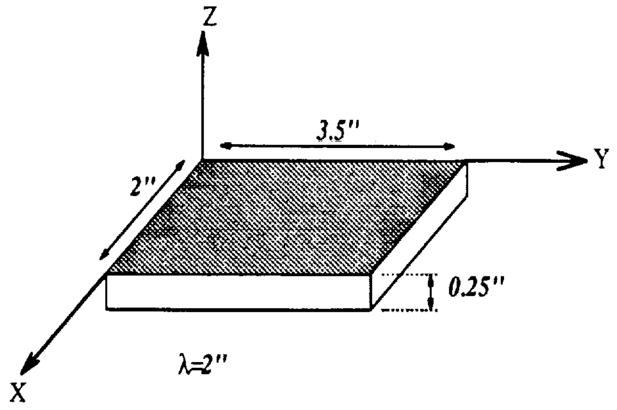
4 Conclusion

In the previous section, we have computed the scattering from geometries of various shapes and material compositions to verify the performance of the conformal boundary conditions that we proposed in the second section. On the basis of the examples shown, it can be safely concluded that the conformal boundary conditions perform quite well in comparison to boundary conditions employed on spherical mesh terminations. This fact, coupled with the enormous savings in computer resources in the form of storage and solution time, highlights the viability of the FEMATS method in solving large three dimensional problems.

In the course of this research, we have also noticed that it is best to keep the shape of the mesh truncation boundary to be as simple as possible. This implies constructing the mesh truncation boundary to be either a sphere or a cylinder or piecewise planar or a combination of these simple shapes. The power of the combined rectangular and cylindrical termination scheme was aptly demonstrated in the case of the perfectly conducting plate where a piecewise planar boundary failed to account for the edge effects but a box-cylinder termination gave significantly better results.

5 References

1. A.F. Peterson, "Absorbing boundary conditions for the vector wave equation", *Microwave and Opt. Tech. Letters*, vol. 1, pp. 62-64, April 1988.
2. J.P. Webb and V.N. Kanellopoulos, "Absorbing boundary conditions for finite element solution of the vector wave equation", *Microwave and Opt. Tech. Letters*, vol. 2, no. 10, pp. 370-372, October 1989.
3. C.T. Tai, *Generalized vector and dyadic analysis*, IEEE Press, New York, 1992.
4. D.S. Jones, "An improved surface radiation condition", *IMA Jour. Appl. Math.*, vol. 48, pp. 163-193, 1992.
5. C.H. Wilcox, "An expansion theorem for electromagnetic fields", *Comm. Pure Appl. Math.*, vol. 9, pp. 115-134, May 1956.
6. S.M. Rytov, "Computation of the skin effect by the perturbation method", *J. Exp. Theor. Phys.*, vol. 10, pp. 180, 1940. Translation by V. Kerdemelidis and K.M. Mitzner.
7. A. Chatterjee, J.M. Jin and J.L. Volakis,"Application of edge-based finite elements and ABCs to 3-D scattering", *IEEE Trans. Antennas Propagat.*, vol. 41, pp. 221-226, February 1993.
8. Courtesy of Northrop Corp., B2 Division, Pico Rivera, CA.
9. H.T.G. Wang, personal communication.



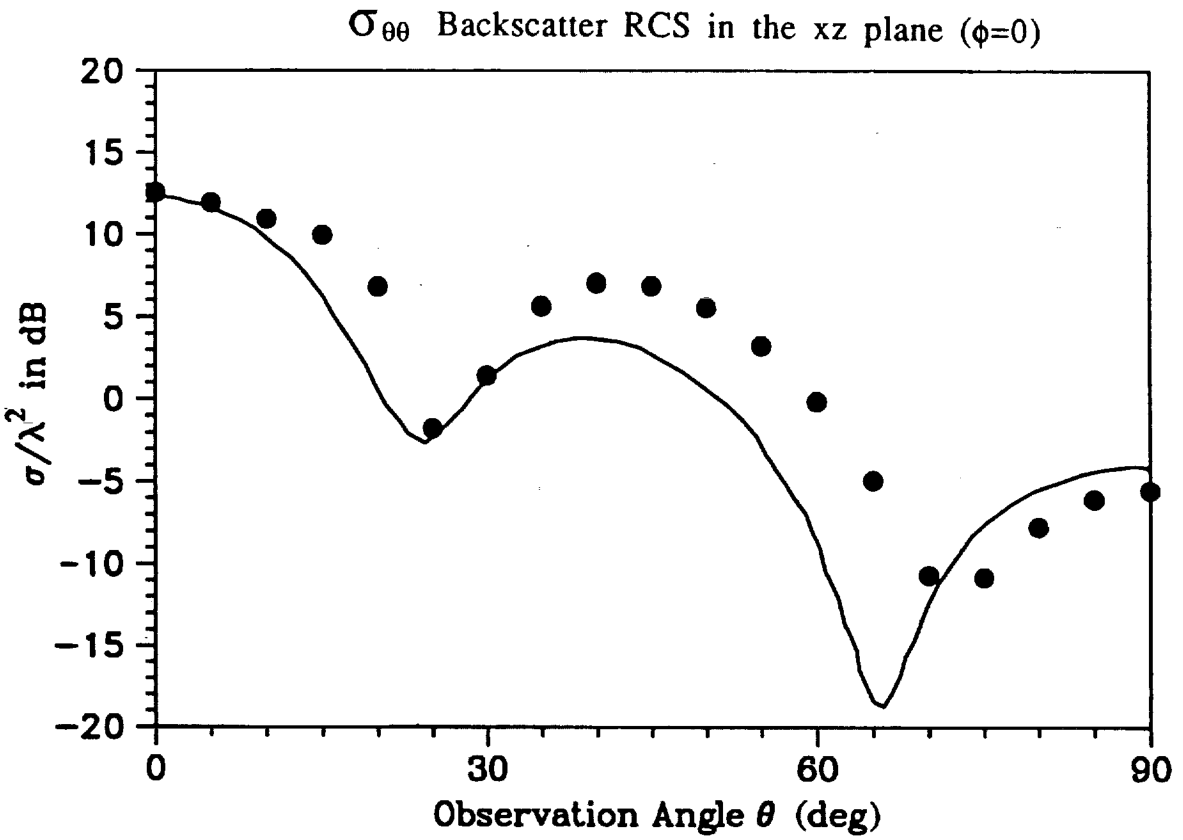


Figure 14: RCS pattern of a glass plate (ϵ_r=5.889 -j.0551; $1\lambda \times 1.75\lambda \times .25\lambda$ at 5.9GHz) for VV polarization on the z-x plane. The solid line denotes moment method data [9] and the black dots indicate computed values.

Parallelization of FEMATS for the Intel iPSC/860

John Nguyen

Introduction

Code for distributed-memory multiprocessors differs from that for conventional sequential processors in several respects. First, taking advantage of the additional processors require that the computations in a program be distributed among the processors. Since most of the work in scientific programs occurs in DO loops, a large amount of parallelism can be achieved by allowing all processors to execute each DO loop. Second, since memory is no longer a contiguous chunk, supporting a distributed-memory hierarchy requires splitting of data into sections for each memory module. Thus the conversion of sequential code to parallel code for the Intel iPSC/860 can be specified as two primary tasks:

1. parallelization of DO loops: Parallelism is introduced in this phase by allowing each processor to execute a portion of each DO loop.

2. distribution of arrays among processors: Since the amount of memory on each processing node is limited, each array is divided into smaller units that reside on each node. This also allows array accesses from each processor to be serviced by different nodes and thereby reduces contention for resources of any single node.

On a cache-only-memory machine such as a KSR, only the first step is necessary since the hardware cache system automatically takes care of the distribution of data among processors. Since the distribution of arrays is the more involved task, this allows one to port a piece of code to the KSR much more quickly than a purely message-passing machine such as the iPSC/860. However, the increased control of data distribution and communication on the iPSC/860 can translate into improved performance for some applications.

The following sections present more detail on the two parallelization tasks as well as some performance figures for the FEM-ATS code.

Parallelization of loops

A DO loop can be completely parallelized when there are no dependences between iterations of the loop. In other words, when each iteration of a loop can be executed independently, then a DO loop can be executed on several processors merely by letting each processor execute a portion of the iterations. For example, a 1000-iteration loop can be parallelized on a 10-processor machine with processor 1 executing iterations 1 through 100, processor 2 executing iterations 101 through 200, etc. In the FEM-ATS code, this approach can be used to parallelize the linear solver as well as the excitation vector generation. However, the main loop in the matrix assembly phase contains a dependence between loop iterations, and thus requires additional modifications.

The primary loop in the matrix assembly phase involves the computation of several intermediate values which are then used to update the matrix. Although the intermediate value computation can be executed independently in each iterations, the matrix update can involve potential conflicts if two processors try to update the same matrix element simultaneously. The parallelization of this loop thus requires an additional mechanism for locking out other processors while updating a matrix element. On the iPSC/860, this is done by allowing each processor to lock a row of the matrix while performing an update. Since the locking of each row is maintained by the processor whose memory holds the row, processors perform row locking and unlocking by sending messages to the appropriate owner of the row.

Distribution of arrays

Whereas the parallelization of loops enables programs to execute faster on multiprocessors, the distribution of arrays must be done for the FEM-ATS program to execute at all. As an example, consider running the FEM-ATS code with 100,000 unknowns. Since the average bandwith of the coefficient matrix is approximately 15, the matrix contains around 1.5 million elements. Each element is a 16-byte complex number which implies that the matrix requires 24 million bytes of storage. Since each iPSC/860 node holds 8 million bytes, the matrix must be distributed before the program can be run at all.

Arrays are distributed in the FEM-ATS code by partitioning one dimension among processors. Thus for a 1000 element array, processor 1 holds the first 100 elements, processor 2 holds the second 100 elements, etc. The straightforward method for accessing this distributed array involves the translation of array references into subroutine calls. Thus an expression `x=a(i)` is translated into the call `call fetcha(i,x)`. The subroutine `fetcha` sends a message to the processor that holds element `a(i)`, which in turn sends a reply message with the value of `a(i)`. Although this scheme requires the implementation of a new subroutine for each distributed array and the replacement of each array access with a subroutine calls, the process is easy and mechanical. However, such a scheme does not result in very good performance, as explained below.

On most multiprocessors, the overhead for sending a message is typically much higher than that of sending a single byte. The cost for sending 10 or even 100 bytes is usually not much higher than that of sending 1 byte. Thus an important strategy in improving performance involves the "bundling" of messages. Rather than sending 100 messages to retrieve 100 bytes, it is much more efficient to send 1 message to retrieve 100 bytes, if possible. The above simple scheme for accessing distributed arrays directly contradicts this strategy. For parts of the code that are not time-critical such as the matrix assembly or excitation vector generation, the simple scheme is adequate. However, since the program spends over 90% of its time in the solver, that part must use a better scheme to access distributed arrays.

The primary operation in the solver that generates communication involves the multiplication of a sparse matrix by a vector. The vector is distributed, as are the rows of the matrix. Since the matrix-vector multiplication involves performing a dot-product of each row with the vector, each processor must obtain the values for the entire vector from other processors. No communication is required for the matrix since each processor already holds the relevant rows of the matrix in its memory. Since each processor may not be able to hold the entire vector in its memory, the dot-product operation must proceed in several phases.

Each processor p begins the matrix-vector multiply by sending its portion of the vector to every other processor, then performs the following for each processor p': Processor p reads the portion of the vector owned by p', and updates the partial dot product for each row by adding the product of each appropriate matrix element with elements of the partial vector. After reading and updating values for all processors p', the dot product operation is complete. Unfortunately, each phase requires a pass over all the sparse-matrix rows owned by the processor in order to find elements that can be multiplied to the current partial vector. In the future, it may be possible to sort each row of the matrix to allow the phases to pass over the rows in order.

Because of the relatively small amount of memory on each iPSC/860 node, each processor can at the moment run the FEM-ATS code on up to 10,000 edges. Thus a run on 40,000 edges would require 4 processors and one on 300,000 edges would require 30 processors. If one were willing to sacrifice some execution speed, this number can be increased by 50% by eliminating some storage that is currently used to reduce the amount of communication between processors.

Results

Results on a problem with 31,000 edges show that the problem scales reasonably for small numbers of processors. However, as the number of processors increase, the problem becomes too small to benefit much from the increase since each processor spends a higher percentage of time on communication and bookkeeping than on true computation. The results are given in number of seconds per iteration as follows:

Number of processors	iPSC/860	KSR
4	.393	.197
8	.290	.104
16	.243	.059
32	.238	

MANUAL OF THE FEM-ATS CODE USED FOR COMPUTING THREE-DIMENSIONAL SCATTERING

Arindam Chatterjee, John L. Volakis
and Mike Nurnberger
Radiation Laboratory
Department of Electrical Engineering
and Computer Science
University of Michigan
Ann Arbor MI 48109-2122

1 Introduction

The FEM-ATS program incorporates first order edge-based finite elements and vector absorbing boundary conditions into the scattered field formulation for computing the scattering from three-dimensional geometries. The code has been validated extensively for a large class of geometries containing inhomogeneities and satisfying transition conditions(see [1] for formulation). The FEMATS code has been optimized to run on the Cray YMP and parallelized to run on the Kendall Square Research architecture and the Intel iPSC/860.

2 Installation

FEMATS is designed to run on multiple computing platforms to best utilize various machine capabilities. Because of the large amount of time required to run FEMATS, it has been written to run on a supercomputer, while I-DEAS and most of the preprocessing programs only need to be run on a unix workstation. Hence there are two sets of source code included on the tape, along with two installation procedures. Also included is a small sample session, starting with the I-DEAS universal file, and ending with the FEMATS output files.

Note: While FEMATS was designed to run on a supercomputer, it may also run on the same workstation that performs the preprocessing, or any other machine.

Installation Instructions

1. Place the distribution tape in tape drive. If the drive is not the default system drive, you will need to find out what device it is.

2. Retrieve the files from the tape to the appropriate directory. If, for example, you wanted to install FEMATS in your homedir, you would say:

   ```
   cd  homedir
   tar xv * .
   ```

   This will place three files in your home directory:

   ```
   femats.reg.tar.Z
   femats.mpp.tar.Z
   femats.ex.tar.Z
   ```

   femats.reg.tar.Z is a compressed tar file containing the source code for the workstation-based portion of FEMATS.

   femats.mpp.tar.Z is a compressed tar file containing the source code for the supercomputer-based portion of FEMATS.

femats.ex.tar.Z is a compressed tar file containing an example run of FEMATS, and may be placed on either system.

3. ftp the supercomputer portion of FEMATS to the supercomputer, putting it in the appropriate directory. (If you are going to run both sections of code on the same machine, don't do this...)

4. If femats.reg.tar.Z is not in the directory where you want to install FEMATS, then put it there. Note: When the files are untared, a directory named femats will be created, and the appropriate files placed in it.

5. Uncompress femats.reg.tar.Z: type

   ```
   uncompress femats.reg.tar.Z
   ```

6. Untar femats.reg.tar: type

   ```
   tar xvf femats.reg.tar
   ```

7. Change directories to femats, and type install.reg. This will compile the preprocessors, and place them in the femats directory.

8. Follow the same steps for the supercomputer, starting with step 4, and changing femats.reg to femats.mpp in all cases.

9. If any problems occur, don't hesitate to look in the scripts—they are quite simple, and there may be some machine dependencies that were missed...

3 Data Generation

The computation of scattering from a specific geometry with FEMATS is a multi-stage process, as is shown in Figure 1. Once the geometric parameters of the target are known, a solid model is constructed in the Solid Modeling family of SDRC I-DEAS, a commercial CAD/CAE/CAM software package. The solid model is then exported to the Finite Element Modeling and Analysis family of I-DEAS, and the nodes and elements necessary for the scattering analysis are generated. This data is written to a output file, called a Universal file, and operated on by several preprocessors, generating the necessary input files for FEMATS.

The process of object modeling and mesh generation is an art, *not* a science. Hence, it cannot be taught, or demonstrated—it must be learned through experience. Therefore, this section is not by any means an I-DEAS FE mesh generation manual. In fact, it assumes (and requires) a working knowledge of, and familiarity with, the I-DEAS Solid Modeling and Finite Element Analysis families.

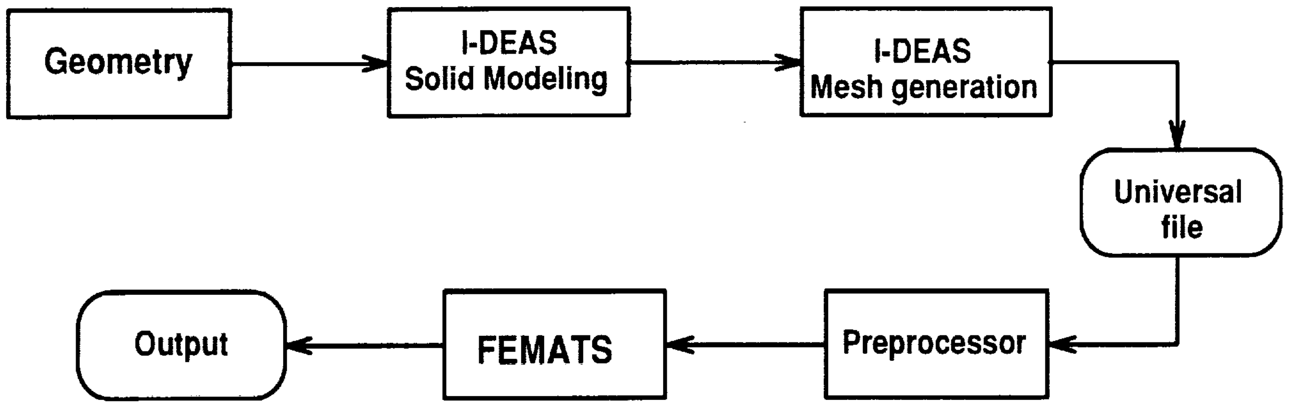


Figure 1: Stages involved in scattering computation from arbitrary 3D geometries

3.1 Solid Modeling

Once the geometry of the target is specified, it is constructed using the I-DEAS Solid Modeling Family of tasks. There is a tendency to downplay the importance of the solid model, and treat it only as a stepping stone towards a final product. However, the solid model is the framework for the finite element mesh, and as such, has a direct bearing on the quality of the mesh. Because of this, and because mesh generation is by no means a science, it is wise to keep the problem as simple as possible. This helps to ensure a better mesh, and a more accurate answer.

In general, the object or body being meshed will contain various planes of symmetry. It is nearly always advisable to take advantage of whatever symmetry is available, as doing so will greatly reduce the amount of time necessary to generate the mesh. In fact, the geometry may require such subdivision to make meshing possible. For more details about the creation of the solid model for FEMATS, please see the I-DEAS Solid Modeling User's Guide.

Note: When creating the Solid Model, FEMATS requires the dimensions to be in wavelengths.

3.2 Mesh Generation

I-DEAS generates the finite element mesh by creating mesh areas on surfaces, and then combining these mesh areas into mesh volumes. Each mesh volume is then filled with the chosen element type. When the solid model is imported into the Finite Element Modeling and Analysis Family of I-DEAS, these mesh areas and mesh volumes are automatically created. Generally, however, I-DEAS does not choose the correct element order (linear vs. parabolic), and the mesh volumes must be modified to reflect the correct element order. Even if the guidelines mentioned above are followed, the mesh areas can become quite complex. It is then prudent to break the mesh volume into smaller, more manageable mesh volumes. For more details on mesh creation, please see the I-DEAS Finite Element Modeling User's Guide.

3.3 Modifying material property labels

After all the mesh volumes have been created , the material property labels of each are modified according to the type of material each mesh volume contains. The elements in the volume between the target and the outer boundary usually has a material property label of 1. If the target contains a dielectric-filled volume, the material property labels of the elements in that volume should be 2 or any integer greater than 1. In this way, the code can accommodate for upto 9 different material fillings. If the geometry should require more than 9 different materials, the vectors *eps* and *mu* should be increased to the required value.

Please see Section 6.1 for more details.

3.4 Type of meshing

The global element length is now specified (usually .075–.085 units); finer meshing can be done in regions with rapidly changing fields or large curvatures by specifying the local element length or curvature-based size parameters. The geometry is then *free*-meshed using the I-DEAS Mesh Creation Module. It is essential to use *free* meshing and not *mapped* meshing since the latter maps the mesh volume into a rectangular box and back, thus distorting the elements. No such distortion occurs in space when an electromagnetic wave travels through it; a *mapped* mesh, therefore, alters the physics of the problem and leads to inaccuracies in the final result.

Please see Section 6.1 for more details.

3.5 Grouping nodes

The finite element method essentially solves a boundary value problem and thus it is crucial to identify surfaces or surface edges on which the boundary conditions are to be imposed. In the current version of FEMATS, this is carried out by grouping the nodes which lie on the surfaces on which the boundary conditions need to be imposed. If the surface nodes coincides with a perfect electric conductor, the group is labeled C, if the nodes lie on a resistive sheet, the group is labeled R and so on. Detailed information about grouping nodes is given in the Appendix.

Care must be taken while grouping surfaces that intersect since edges connecting two such nodes may not lie on the surface at all. For example, three surfaces intersect at the corner of a cube. If the nodes on each of these surfaces are not grouped separately, the processing program will give rise to 'surface' edges which actually do not lie on the surface. Another anomaly may arise when the surfaces are separated by a single element. This is due to the fact that the processing program considers an edge to lie on the surface if two nodes in the group connect to form a edge. As a rule of thumb, it is best to group each surface separately. They may be grouped together only when the user is certain that spurious surface edges will not be created by the processing program.

Please see Section 6.1 for more details.

3.6 Universal file

The mesh information obtained from I-DEAS is now written to an ASCII file called the Universal file. The Universal file has a specific format for identifying the nodes, elements and groups which can be obtained from the I-DEAS User's guide.

It should be noted that only the FE entities and Groups need to be included into the Universal file. Also, while this discussion has dealt primarily with I-DEAS, any mesh generator that writes a Universal file will work just fine...

4 Preprocessing

The necessary preprocessing is performed by a number of smaller programs that operate on the Universal file generated by I-DEAS. Because FEMATS is designed to be run on a supercomputer, and I-DEAS and the preprocessors generally are run on a workstation of some sort, some of the preprocessing is performed on the workstation, and some of it on the supercomputer. In both cases, a script runs the necessary preprocessors, and presents the user with the necessary FEMATS input data files. (Note that while FEMATS has been designed to run on a supercomputer, it can peacefully co-exist with the preprocessors, etc. on the workstation. However, if FEMATS is to be run on the workstation, certain variable types must be changed to reflect the decreased precision inherent to most workstations.) To run the script that runs the preprocessors, type

```
femats.reg file.unv
```

where file.unv is the name of the universal file containing the mesh information. The femats.reg script extracts the necessary information from each preprocessor, and terminates with instructions informing the user which files need to be transfered to the supercomputer for further preprocessing.

After the appropriate files have been transfered to the supercomputer, a script is also run there to finish the preprocessing. To run this script, type

```
femats.mpp file
```

where file is the name of the original universal file, *without* the ending (.unv). This script will present the user with the necessary input files for FEMATS, along with a list of numbers required for input by FEMATS.

5 Running FEMATS

```
Enter 1 for data to be entered interactively; 2 for
data to be read from a file
```

It is faster to get data read from a file; however, for the first-time user, interactive input provides more insight.

```
Number of edges
```

Input the no. of edges obtained from **proc**.

```
No.\ of elements with surface edges on 1)pec 2)r-card 3) ibc
4)dielectric 5)outer boundary 6)outer surface of scatterer
```

Enter the no. of elements with surface edges on the various materials as obtained from **proc.f**.

```
If inhomogeneous, enter 0
```

Enter 0 as long as there are two or more material property labels (see Appendix) present in the geometry. This holds for r-cards as well, since the top and bottom elements on a r-card have different material property labels.

```
Number of distinct dielectric materials
```

Enter the no. of distinct material property labels. Note material property label 1 is free-space by default. If the mesh designates material property label 1 to anything other than free-space, the program won't run.

```
constitutive relative parameters for region ,i
```

Input the epsilon and mu of the dielectric in that order. For a r-card whose top and bottom surface is free-space, enter ϵ_r and μ_r of free space, i.e., unity.

```
If resistive card inside geometry enter 1

Number of different r-cards
```

Input the no.of r-cards having different resistivity values for the geometry.

```
Input: a) Material property label on top surface of card
       b) Material property label on bottom surface of card
       c) Normalized resistivity
```

The mesh must be constructed in such a way that the material property labels on the top and bottom surfaces of the R-card are different.

```
If impedance sheet inside geometry enter 1

Input: a) Material property label on top surface of impedance sheet
       b) Normalized impedance
```

Most of the data entered till now has been related to the geometry. The data entered from this point will be related to the iteration count, number of look angles, etc.

```
Tolerance, maximum iterations
```

The tolerance of the residual is usually kept between .001 and .0005. This is .1%–.05% of the solution norm. Max. no. of iterations is determined by trial and error. A typical value for PEC targets is $N/100$ for $N > 25000$ and $N/120$ for $N > 75000$. The largest problem run to date contained 93000 unknowns and converged, on the average, in 800 iterations. The code uses diagonally preconditioned biconjugate gradient method to solve the system so the residual error will jump to abnormal values quite frequently.

```
1) Bistatic  2) Backscatter
```

Enter 1 for bistatic pattern, 2 for backscatter

```
All angle values should be integers

Bistatic
--------
Angle of incidence: theta,phi
Fix 1)phi 2)theta to specified angle
Angle of observation: start,end,increment
Polarisation angle: alpha=0(H_z=0); alpha=90(E_z=0)
```

In order to fix ϕ to 90° (say), the input should look like
1 90
To fix θ to 90° (say), the input should be
2 90

```
Backscatter
-----------
Fix 1)phi 2)theta to specified angle
Angle of incidence: start,end,increment
Polarisation angle: alpha=0(H_z=0); alpha=90(E_z=0)

Enter 1 for spherical outer boundary; 2 otherwise
```

The code works for a spherical termination or terminations having flat outer faces. The sphere should be centered at the origin.

6 Appendix

6.1 Stipulations for mesh generation

- the region surrounding the scatterer should have a material property number label of 1, i.e., the least possible value.
- for a surface draped by a resistive card, it is essential to differentiate the top surface from the bottom surface. The only way the program can discern this from the available data is by checking the material property number labels of the elements on the top and bottom surfaces. The material property number label of the top surface must be different from that of the bottom surface.
- when meshing a mesh-volume filled with a dielectric having a certain permeability and permittivity, the material property label number should be different from that of surrounding space.
- when grouping surface nodes, the group labels should start with a

  - **C** if the nodes lie on a perfect electrical conductor
  - **R** if the nodes lie on a resistive card
  - **D** if the nodes lie on a dielectric
  - **A** if the nodes lie in free space (i.e., on the mesh termination boundary)
  - **O** if the nodes lie on the outer surface of the scatterer

  The above order (C,R,D,A,O) <u>must</u> be maintained when grouping nodes.

- Nodes on the interfaces of materials having different constitutive parameters must be grouped.

6.2 Code Theory of Operation

6.2.1 proc.f

The program converts the mesh information stored in the Universal file into a more usable form for analysis by FEMATS. It first reads in the nodal coordinates, nodal connectivity and the grouped nodes from the Universal file. Since FEMATS uses edge-based shape functions, the edges and the nodes connecting them need to be identified. Since each edge is shared by more than one element, care must be taken so that the same edge is not counted more than once. A comparison of the connecting nodes must therefore be done for identifying old edges and creating new ones. This can be a computationally intensive task if a brute force comparison is carried out, especially if the problem size is very large. We need to use an algorithm that would scale at most linearly with the number of nodes or edges, i.e. the number of comparisons required for identifying old or new edges should be an $O(N)$ process.

In order to realize this requirement, we use the ITPACK scheme [2] for storing the node connectivity information. The ITPACK scheme is attractive since the number of comparisons required while augmenting the connectivity matrix depends only on the locality of the corresponding node and not on the total number of nodes or edges. In the ITPACK storage scheme, the no. of rows of the connectivity matrix is equal to the no. of nodes and the no. of columns equals the maximum no. of nodes connected to a particular node. Since this leads to a wastage of space when the no. of connecting nodes varies widely, we use a modified ITPACK format where the no. of columns of the connectivity matrix equals the average no. of nodes connected to a particular node and the no. of rows is slightly more than the total no. of nodes. The storage requirement for such a matrix is usually $1.1N_n \times 16$ integers, where N_n equals the no. of nodes.

After generating the edges, the code uses the same storage scheme for finding the surface edges and elements from the grouped nodes. The surface edges are then sorted in ascending order by element number for the various materials and boundaries on which they lie. All components of the code are extremely fast with the slowest being the sorting routine.

The output files from *proc.f* are

- **enode**
  contains co-ordinates of all the nodes in the geometry.
- **eglob**
  contains the edges making up each element.
- **edge**
  contains the nodes making up each edge.
- **esurfed**
  contains the element number, node numbers nad corresponding edge numbers of the on-surface edges.
- **otpt**
  contains the number of edges in the geometry and the number of elements with surface edges on the PEC, R-card, dielectric, outer boundary and outer surface of scatterer.

Required storage is about $18N$ real Words, where N is the number of unknowns and is equal to the number of edges making up the mesh.

6.2.2 count.f

The program asks for the number of edges in the geometry and generates **cntr** as the output. **cntr** contains the no. of non-zero entries per row for the finite element system. The number usually varies from 9 to 31 for a typical system. Required storage is about $13N$ real Words, where N again denotes the number of unknowns.

6.2.3 fem.f

This is the main program (FEMATS) which computes backscatter or bistatic patterns after reading in the mesh files created by *proc.f* and *count.f*. Parameters like number of edges, number of surface elements, type of pattern, etc. can be read in interactively or from a file. The backscatter or bistatic pattern is returned in a separate file. If the code fails to run for some reason, the error is returned in the error file. The flow of control of FEMATS is given in Figure 2. The formulation for the methodology is given in [2].

Input files: The input files containing the mesh information and parameters for running the probelm are read in, usually in binary format. The ASCII format is quite slow for most machines and prohibitively slow on the KSR1. A small program usually converts the mesh files from ASCII to binary.

Processing data: Some preliminary processing is done to find the radius of the outer boundary if a spherical mesh termination scheme is used.

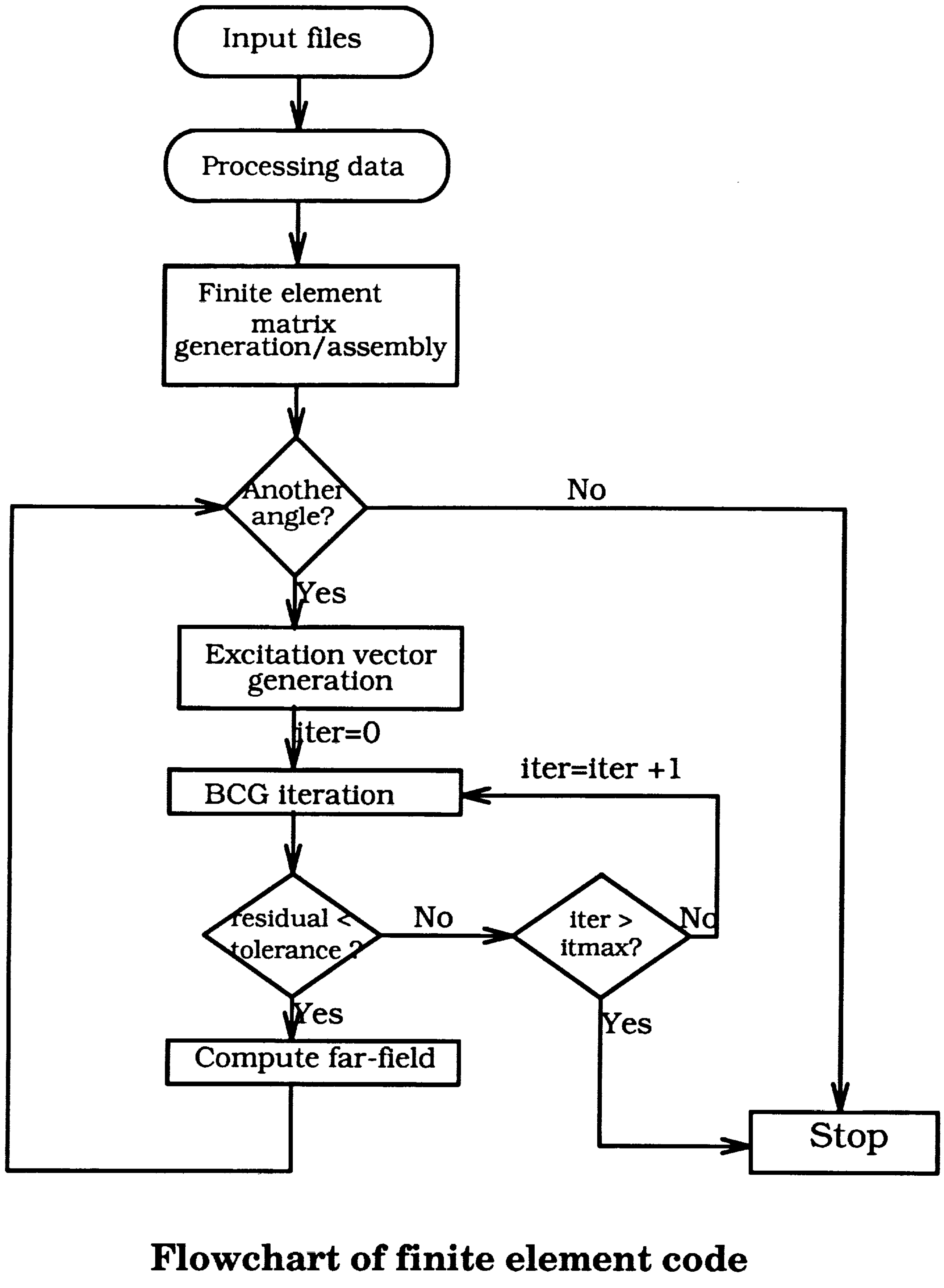


Figure 2: Flowchart for FEMATS

FE matrix generation/assembly: The finite element matrix generation is done on an element-by-element basis. The elemental matrix is first computed and then assembled into the global sparse matrix. The assembly is simplified since the number of non-zero entries per row of the matrix is known *apriori* and the order of the entries is not important. The non-zero entries of the final sparse matrix are stored in a long complex vector, the corresponding column nos. are stored in an integer vector and the location of the first non-zero entry for each row is stored in another integer vector. This is the well-known Compressed Sparse Row (CSR) format used in public domain software packages like SLAP and SPARSPAK. The coefficient matrix is not a function of the angle of incidence.

The code also uses a simple diagonal preconditioner for speeding up the iterative process. Other complicated preconditioning strategies are also available. However, except the block ILU preconditioner, none of them compare favourably with the point diagonal preconditioner in terms of solution time.

Excitation vector generation: The excitation vector generation is not very computer intensive since the vectors are always quite sparse. It is a function of the angle of incidence.

BCG iteration: The biconjugate gradient (BCG) algorithm is used with preconditioning to solve the sparse, symmetric system of linear equations. Each iteration of the algorithm involves 1 sparse-matrix vector product, 3 vector updates and 3 inner products. The norm of the residual vector is computed after every iteration to check for convergence. Reliable results have been obtained by setting the convergence criterion to be

$$||r^k|| < .001 * ||b||$$

where r^k is the residual vector after the kth iteration and b is the excitation vector.

Far-field evaluation: The far-field is evaluated by integrating the near-zone fields over a closed surface using the Stratton-Chu integral equation. The surface is usually taken to be very close to or on the body itself to achieve maximum accuracy.

Storage required for the code is at present $36N$ complex Words, where N is the number of unknowns. The storage can be cut by 40% if only the symmetric upper triangular part of the object matrix is stored; the code, however, slows down significantly.

6.2.4 Subroutine functions in fem.f

basis.f

Calculates the two constant vectors of the bases for the finite element discretization as well as the element volume.

calc.f

Computes the volume integral for the finite element discretization analytically.

ccross.f
Takes the cross product of two complex vectors.

cdot.f
Takes the inner product of two vectors.

comput.f
Calculates the basis functions at the mid-point of each edge.

cross.f
Takes the cross product of two real vectors.

crux.f
Computes the element matrix from the volume integral.

cruxd.f
Imposes the boundary condition for dielectric volumes and generates the corresponding excitation vector.

dist.f
Calculates the distance between two points in space.

dot.f
Computes the dot product of two real vectors.

exchg.f
Exchanges one variable with another.

fl.f
Carries out the volume integration of $W_i \cdot W_j$ analytically.

finc.f
Computes the volume integral for a dielectric volume to be used in the excitation vector.

fu.f
Carries out the surface integration for the absorbing boundary condition employed on the mesh termination boundary.

incc.f
Imposes the boundary condition for a perfect electric conductor. If *iter* is 0, the excitation vector is computed, otherwise changes are made to the element matrix.

incd.f
Imposes the boundary condition for a dielectric surface.

incr.f
Imposes the boundary condition for a resistive card.

mult.f

Carries out the sparse matrix-vector multiplication.

norm2d.f

Computes the element normal for a 2D geometry.

norma.f

Computes the element normal for a surface element.

ord.f

Identifies the global nodes and edges in the local context.

sort.f

Sorts the edges in a element according to a specific numbering scheme.

surfint.f

Imposes the absorbing boundary condition on the mesh termination boundary.

value.f

Computes the far-field using the Stratton-Chu integral equation.

volume.f

Calculates the element volume.

6.2.5 References

1. A. Chatterjee, J.M. Jin and J.L. Volakis, "Application of edge-based finite elements and ABCs to 3-D scattering," *IEEE Trans. Antennas Propagat.*, vol. 41, pp. 221–26, February 1993.

2. D.R. Kincaid and T.C. Oppe, "ITPACK on supercomputers," *Numerical Methods, Lecture Notes in Mathematics*, vol. 1005, pp. 151–61, Springer, Berlin, 1982.

6.3 Example FEMATS Run

A perfectly conducting cylindrical inlet was run on the KSR1 machine. The geometry was enclosed by a rectangular outer boundary and the backscatter pattern was sought for $\theta = 0°$–$90°$ and $\alpha = 90°$. The problem had 213,832 unknowns and a diagonally preconditioned BCG solver was used.

Input file:

```
213832
13656 0 0 10704 7704
0
2
(1.,0.) (1.,0.)
0
eg
.001 10000
2
1 90
0 90 5
90
2
```

Output file:

```
 Number of threads =   56
 Backscatter pattern will be computed
 Polarisation angle=   90
 Incident angle from   0  to   90
 in steps of   5
 Sweep through theta ; phi=   90
 ************************************************
               Problem size
 Number of nodes =   32453
 Number of elements =   176048
 Number of edges/unknowns =   213832
 ************************************************
 Finished reading in data
 Outer boundary shape is
 Rectangular
 Time spent for unformatted I/O =     1.0519631999999999  secs
 Time spent for I/O =     1.0754451999999999  seconds
  10000
 Generating finite element matrix
 Generated finite element matrix
 No.\ of non-zeros =   3414496
 Average no.\ of non-zeros =   15
 Total time spent=     25.834841199999996  secs
 Time spent in loop=     24.807681599999999  secs
 Generated diagonal preconditioner
 Time for preconditioner =    1.1077387999999999  secs
 ************************************************
   90.000000000000000   0.    90.000000000000000
  (   23874.682292021858,  0.)
 Time spent in gen. soln. vector =     19.514778000000000  secs
```

```
Convergence achieved in   4397  iterations
Time spent in   4397  iterations =     584.26175160000003  secs
Backscatter =     13.132205300654515

   90.000000000000000    5.0000000000000009    90.000000000000000
 (   23874.649727818964,  0.)
Time spent in gen. soln. vector =     20.677881599999978  secs
Convergence achieved in   1878  iterations
Time spent in   1878  iterations =     248.19567480000001  secs
Backscatter =     12.346785646714235

   90.000000000000000    10.000000000000002    90.000000000000000
 (   23874.552995090075,  0.)
Time spent in gen. soln. vector =     20.646470399999998  secs
Convergence achieved in   6561  iterations
Time spent in   6561  iterations =     866.82569879999994  secs
Backscatter =     10.984172458513957

   90.000000000000000    15.000000000000002    90.000000000000000
 (   23874.394947730074,  0.)
Time spent in gen. soln. vector =     20.678055200000017  secs
Convergence achieved in   6112  iterations
Time spent in   6112  iterations =     807.46988880000004  secs
Backscatter =     8.0404387189358921

   90.000000000000000    20.000000000000004    90.000000000000000
 (   23874.180257205706,  0.)
Time spent in gen. soln. vector =     20.636231199999656  secs
Convergence achieved in   6430  iterations
Time spent in   6430  iterations =     849.45422520000011  secs
Backscatter =     4.5520666697643231

   90.000000000000000    25.000000000000000    90.000000000000000
 (   23873.915286302414,  0.)
Time spent in gen. soln. vector =     20.640396400000100  secs
Convergence achieved in   6303  iterations
Time spent in   6303  iterations =     832.76715800000011  secs
Backscatter =     1.8286943794696267

   90.000000000000000    30.000000000000004    90.000000000000000
 (   23873.607915069253,  0.)
Time spent in gen. soln. vector =     20.624299199999768  secs
Convergence achieved in   4543  iterations
Time spent in   4543  iterations =     600.40641159999996  secs
Backscatter =     2.1870075445461543
```

```
   90.000000000000000   35.000000000000007   90.000000000000000
 (   23873.267321948337,  0.)
Time spent in gen. soln. vector =     20.654717999999775  secs
Convergence achieved in   6015  iterations
Time spent in   6015  iterations =     794.75217840000005  secs
Backscatter =     2.9538913638132449

   90.000000000000000   40.000000000000007   90.000000000000000
 (   23872.903724276915,  0.)
Time spent in gen. soln. vector =     20.636641200000668  secs
Convergence achieved in   6215  iterations
Time spent in   6215  iterations =     821.00750359999984  secs
Backscatter =     4.3094572190189613

   90.000000000000000   45.000000000000000   90.000000000000000
 (   23872.528083709803,  0.)
Time spent in gen. soln. vector =     20.691929999999957  secs
Convergence achieved in   3213  iterations
Time spent in   3213  iterations =     424.64956839999923  secs
Backscatter =     4.0201582083152863

   90.000000000000000   50.000000000000000   90.000000000000000
 (   23872.151783594894,  0.)
Time spent in gen. soln. vector =     20.679000399999495  secs
Convergence achieved in   3196  iterations
Time spent in   3196  iterations =     422.23548159999973  secs
Backscatter =     6.0710219487833603

   90.000000000000000   55.000000000000000   90.000000000000000
 (   23871.786286832561,  0.)
Time spent in gen. soln. vector =     20.649760399999650  secs
Convergence achieved in   5037  iterations
Time spent in   5037  iterations =     665.48518600000079  secs
Backscatter =     6.0386806852857617

   90.000000000000000   60.000000000000007   90.000000000000000
 (   23871.442784137245,  0.)
Time spent in gen. soln. vector =     20.630159599999388  secs
Convergence achieved in   5096  iterations
Time spent in   5096  iterations =     673.35249359999943  secs
Backscatter =     2.9883959797387871

   90.000000000000000   65.000000000000000   90.000000000000000
 (   23871.131843785708,  0.)
Time spent in gen. soln. vector =     20.649902799999836  secs
Convergence achieved in   5096  iterations
```

```
Time spent in   5096  iterations =     673.46241600000030  secs
Backscatter =     4.1405354898674034

   90.000000000000000    70.000000000000014    90.000000000000000
 (   23870.863074712492,  0.)
Time spent in gen. soln. vector =     20.627787199999148  secs
Convergence achieved in   4893  iterations
Time spent in   4893  iterations =     646.38884879999932  secs
Backscatter =     3.4527505854226375

   90.000000000000000    75.000000000000014    90.000000000000000
 (   23870.644815085496,  0.)
Time spent in gen. soln. vector =     20.643494799998734  secs
Convergence achieved in   3459  iterations
Time spent in   3459  iterations =     457.10817280000083  secs
Backscatter =   -0.42219432577245863

   90.000000000000000    80.000000000000014    90.000000000000000
 (   23870.483858181193,  0.)
Time spent in gen. soln. vector =     20.630811200000608  secs
Convergence achieved in   4885  iterations
Time spent in   4885  iterations =     645.37610479999967  secs
Backscatter =     4.9808095185448629

   90.000000000000000    85.000000000000000    90.000000000000000
 (   23870.385226404902,  0.)
Time spent in gen. soln. vector =     20.668981599999825  secs
Convergence achieved in   1924  iterations
Time spent in   1924  iterations =     254.26892960000077  secs
Backscatter =     8.0244739010739181

   90.000000000000000    90.000000000000000    90.000000000000000
 (   23870.352002710646,  0.)
Time spent in gen. soln. vector =     20.646247599999697  secs
Convergence achieved in   1747  iterations
Time spent in   1747  iterations =     231.11860320000051  secs
Backscatter =     8.7459369327904284

Total time =     11978.956639599999  seconds
```

APPENDIX

A preprocessing algorithm to find boundary surfaces and normals in finite element data structures

Daniel C. Ross
Radiation Laboratory
University of Michigan

Introduction

Electromagnetic analysis with the finite element method can be thought of as a four step process: mesh generation, preprocessing, analysis and then postprocessing. The preprocessing step is often incorporated into the analysis code or considered to be part of the mesh generation step. For large, complex objects (especially in three dimensions) this task can be very cumbersome and should be considered as a separate process.

Preprocessing involves the extraction of information which are required for imposing boundary conditions on metal surfaces, mesh truncation surfaces where some absorbing boundary condition is enforced, as well as material discontinuities and outward normals to these surfaces. For field calculations outside the region of the mesh, an integration contour or surface must also be found that completely encloses the object while passing through element centers where field derivatives can be evaluated accurately for calculating the scattered field. Without an accurate evaluation of the field derivatives on such a surface, substantial error (3dB) in far field calculations, and even more inaccuracy in near field calculations is likely to result. Although all of the above information could be specified during mesh generation, this would involve in general, a great deal of unnecessary work. The goal of the preprocessor should be to minimize the amount of interaction with the user.

The following algorithms have been developed for preprocessing a finite element mesh for electro-magnetic analysis. The mesh is only required to contain a minimum amount of geometric information. The algorithms will extract from the mesh: metal boundaries (any number of metal objects), outer boundary, material boundaries, outward pointing normals, element edges lying on conductors (for edge based analysis) and an integration contour passing through element centers. Although the algo-rithms are appropriate for many different elements in two and three space dimensions, most of the code fragments shown are for the particular case of tetrahedral elements.

The code is written in C since recursion and dynamic memory allocation are needed.

Data Structures

The mesh is specified by filling the following simple structures from the Universal file which con-tains the node locations and element specifications as generated by the mesh generator.

```
int nnodes;                        Number of nodes in mesh
int nelements;                     Number of elements in mesh
struct node {
     float x,y,z;                  Array of node coordinates
} *nodes;
struct element {
      int nodes[4],material;       Array of tetrahedral elements (4 nodes and a material
} *elements;                       index number)
struct material{
     float ReE,ImE,ReM,ImM;        Array of materials (Complex ε and μ)
} *materials;
```

This is enough information to find metal surfaces, the outer surface of the mesh, material discon-tinuities, outward pointing surface normals, element edges lying on metal surfaces and an integration contour that encloses the target but passes through element centers. Nodal grouping is required only to identify infinitely thin conductors or edges and corners that are to be treated differently by the finite element code (singular elements). No more information is needed from the user.

Once the mesh data structures are filled, the mesh tree must be created. The mesh tree is a two dimensional array that stores the element numbers of elements that share a common node. The struc-ture is cheap to fill and allows for the efficient extraction of all the needed information.

This operation fills the mesh tree with all of the elements in the mesh.

```
o
o
o
#define MAXCONNECTIONS 50
int **mesh_tree
int n,m,n1,i;
o
o
o
```

Maximum number of edges sharing a node.
Two dimensional array to store mesh tree.
Some integers.

```
mesh_tree=(int **) calloc(nnodes+1,
  sizeof(*mesh_tree));
for (n=0;n<=nvertices+1;n++)
  mesh_tree[n]=(int *) calloc(
  MAXCONNECTIONS,sizeof(**mesh_tree
));
o
o
o
```

Dynamically allocate memory for the mesh tree and initialize to zero.

```
for (n=1;n<=nelements;n++)
  for (m=0;m<4;m++)
     n1=elements[n].vertices[m];
     i=0;
  while(mesh_tree[n1][i]!=0)
     i=i+1;
     mesh_tree[n1][i]=n;
     mesh_tree[n1][i+1]=0;
  }
```

Fill mesh tree with all elements.

Operations

•Find free surface

This algorithm operates on the mesh tree filled with all elements.

Let n_{face} be the number of coplanar nodes defining an element face. (n_{face}=3 for a tetrahedral element.)

This algorithm efficiently scans the mesh tree for groups of n_{face} coplanar nodes (faces) shared by more than one element. A face which is not shared by more than one element is a free face.

```
#define MAXCONNECTIONS 50                       Maximum number of edges sharing a
                                                node.
struct freeElement{                             Structure for storing free elements (trian-
  int element;                                  gular faces). It contains: the element that
  int nodes[3];                                 the face belongs to, the three nodes that
  int surface;                                  define the face,surface number.
} *freeElements;
int nFreeElements;                              Some integers
int e,i,j,k,n,m,foundflag;
int nFreeElements=0;
int ns[4];
int ecount [3*MAXCONNECTIONS][2];               Common element counter
o
o
o
for (n=1;<=nelements;n++)                       Every element
  for (i=1;i<=4;i++)                            Every face
     switch (i) {                               Get nodes for each face
        case 1;
           ns[0]=elements[n].node[0];
           ns[1]=elements[n].node[1];
           ns[2]=elements[n].node[2];
           break;
        case 2;
           ns[0]=elements[n].node[0];
           ns[1]=elements[n].node[1];
           ns[2]=elements[n].node[3];
           break;
        case 3;
           ns[0]=elements[n].node[1];
           ns[1]=elements[n].node[2];
           ns[2]=elements[n].node[3];
           break;
        case 4;
           ns[0]=elements[n].node[0];
           ns[1]=elements[n].node[2];
           ns[2]=elements[n].node[3];
           break;
     }
     for (j=0;j<3*MAXCONNECTIONS;j++) {         Zero out common element counter
        ecount[j][0]=0;
        ecount[j][1]=0;
     }
     for (j=0;j<=2;j++) {                       Fill common element counter
        m=0;
        while (mesh_tree[ns[j]][m]!=0) {
           e=mesh_tree[ns[j]][m];
           k=0;
           while (ecount[k][0]!=e&&
              ecount[k][0]!=0)
```

```
            if (ecount[k][0]==0) {                Add new element to common element
               ecount[k][0]=e;                    counter
               ecount[k][1]=1;
            }
            else
               ++ecount[k][1];                    Increment common element counter
            ++m;
         }
      }

      k=0;                                        Determine if face is free or shared
      foundflag=0;
      while (foundflag==0) {
         while (ecount[k][1]!=3 &&
            ecount[k][1]!=0)
            ++k;
         if (ecount[k][0]==n) ++k;                Don't count self shared face
         else foundflag=1;
      }

      if (ecount[k][1]<3) {                       Test if free face
         ++nFreeElements;                         Free face
         freeElements[nFreeElements].             Update freeElements (face) data struc-
            element=n;                            tures.
         freeElements[nFreeElements].
            nodes[0]=ns[0];
         freeElements[nFreeElements].
            nodes[1]=ns[1];
         freeElements[nFreeElements].
            nodes[2]=ns[2];
      }
      o
      o                                           The following operations can be put here
      o                                           if needed.
} }
```

Upon completion of this operation there is one free surface. This will be divided into pieces (outer surface, metal surfaces) by filling the mesh tree with the free elements only and doing a divide free surface operation.

• **Find material discontinuities and normals**

Inside the **find free surface** loop, check the material numbers of each element pair sharing a face. If they are different, store the shared face as a free surface and calculate the normal in the same manner as above. Consistent normal directions are found by ensuring that the normal points towards the lower material index.

```
if (elements[n].material < elements[ecount[k][0].material) {
o
o
o
}
```

• **Find outward normals to free surface**

To ensure that all free face normals have the correct sign, the following inexpensive test will guarantee that all normals are pointing outward.

Given an element with a free surface defined by the four nodal points (x_i, y_i, z_i), i=1,2,3,4

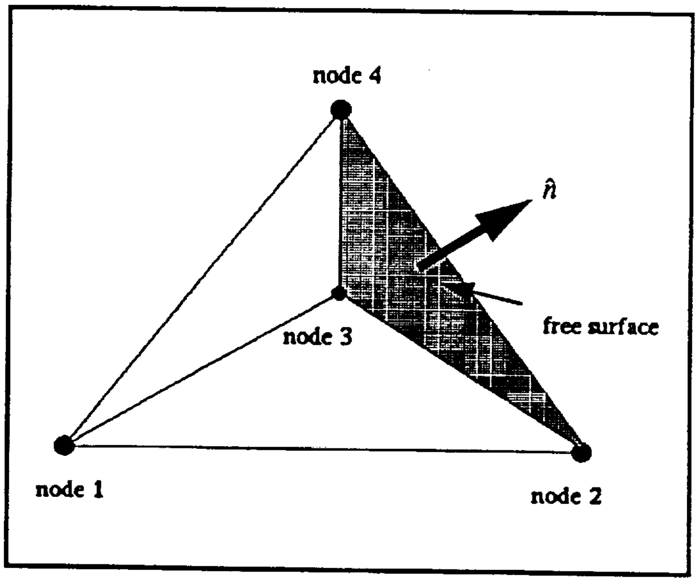


FIGURE 1. Example element

the normal (to within a sign) is given by

$$\hat{n} = \frac{(\vec{r}_2 - \vec{r}_3) \times (\vec{r}_4 - \vec{r}_3)}{|(\vec{r}_2 - \vec{r}_3) \times (\vec{r}_4 - \vec{r}_3)|}$$

where

$$\vec{r}_i = \hat{x}x_i + \hat{y}y_i + \hat{z}z_i$$

To find the correct sign, compare the distance from the node not on the free surface (node 1 for the example shown in figure 1) to a point that is a unit length along the normal in either direction. The direction giving the larger distance is the correct one.

•Divide free surface (find metal surfaces and outer surface of mesh)

This algorithm divides the entire free surface found by the **find free surface** operation into disconnect pieces. To accomplish this, the mesh tree is filled with all of the free elements found by the **find free surf** operation and is recursively scanned to find groups of connected elements. Each surface element is tagged with a surface number and the volume of each surface is compared. The surface with the largest volume is then tagged as the outer surface and all other surfaces are tagged as metal surfaces. There can be multiple, unconnected metal bodies in the mesh.

```
for(n=1;n<=NFreeElements;n++)                          Fill mesh tree with free surface elements
     for(m=0;m<3;m++) {                                only.
         n1=FreeElements[n].4;
         i=0;
         while( mesh_tree[n1][i]!=0) i=i+1;
         mesh_tree[n1][i]=n;
         mesh_tree[n1][i+1]=0;
         }
surface=1;
while(surface!=0) {                                    Find connected free elements by recursive
      i=1;                                             scanning the mesh tree.
      j=0;
      foundflag=0;
      while(foundflag==0) {
         while(mesh_tree[i][j]== -1) j=j+1;            Drop out when mesh tree is empty
         if (mesh_tree[i][j]==0)
           if (i <= nvertices) {
             i=i+1;
             j=0;
         }
         else {
           foundflag=1;
           surface=0;
         }
         else {
           foundflag=1;
           e=mesh_tree[i][j];
         }
      }
      if (surface!=0) {
        error=update(e,mesh_tree,                      Call to recursive function for updating mes
        FreeElements,surface);                         tree. (See below)
        nsurfaces=surface;
        surface=surface+1;                             Keep count of distinct surfaces.
      }
}
```

```
int update(e,mesh_tree,FreeElements,           Recursive function to remove connected
surface)                                       free elements from mesh tree.
struct FreeElement{
 int element;
 int FreeNodes[1];
 int FreeVertices[3];
 int surface;
};
int e,surface;
struct FreeElement *FreeElements;
int **mesh_tree;
{
  int n[3],j,m,f,i,error;
  n[0]=FreeElements[e].FreeVertices[0];
  n[1]=FreeElements[e].FreeVertices[1];
  n[2]=FreeElements[e].FreeVertices[2];
     for (i=0;i<3;i++) {
     j=0;
     m=n[i];
     while(mesh_tree[m][j]!=0) {
          f=mesh_tree[m][j];
          if (f != -1) {
            FreeElements[f].surface=
            surface;
            mesh_tree[m][j] = -1;
            error=update(f,mesh_tree,        Recursive call
            FreeElements,surface);
          }
          j=j+1;
     }
  }
  return(1);
}
```

Now that the free surface has been split into pieces, the surface with the largest volume must be the outer surface and all others must be metal surfaces. To calculate the volumes of the surfaces, add up the contributions from each free face as

$$V = \sum_{e=1}^{N_{surf}} \left| (\vec{v}_1 \times \vec{v}_2) \cdot \vec{r}_0 \right|$$

where

$$\vec{v}_1 = \hat{x}(x_3 - x_2) + \hat{y}(y_3 - y_2) + \hat{z}(z_3 - z_2)$$
$$\vec{v}_2 = \hat{x}(x_4 - x_2) + \hat{y}(y_4 - y_2) + \hat{z}(z_4 - z_2)$$
$$\vec{r}_0 = \hat{x}(x_2) + \hat{y}(y_2) + \hat{z}(z_2)$$

and (x_i, y_i, z_i) are the nodal coordinates of the element having a free face (see figure 1) and N_{surf} is the number of free faces in the surface.

• Find edges lying on a conductor

Now that all of the free surfaces have been found, each free face belonging to a metal surface (all surfaces except the surface with the largest volume) can be split into three edges. This information is necessary for edge-based finite element analysis.

• **Find enclosing integration contour**

This operation finds a surface (three-dimensions) or contour (two-dimensions) that completely encloses the target and passes through element centers. This routine relies on the previous ones since it requires the divided free surfaces and material interfaces.

The mesh tree must be loaded with all elements (not just free surface elements). The tree is then scanned for elements having the following properties:

1 - The element's material is air (or surrounding media) and has one, two or three nodes lying on a metal surface or a dielectric interface.

2 - The element does not have more than one face on either a metal surface or a dielectric interface.

Requirement 1 makes sure that all of the scatterer (metal and dielectric) is enclosed. Requirement 2 ensures that the surface/contour will be closed and relatively smooth.

Any element meeting these requirements is divided by extracting all its vertices that have one and only one node on a metal surface or dielectric interface. A new node is created at the mid point of each vertex and a new face is created from these points. Taken collectively, these new faces make up a closed surface/contour that completely encloses the scatterer and passes through element centers where field derivatives can be calculated accurately.

In figure 2, part of the integration contour for a two-dimensional coated conductor with a crack is shown.

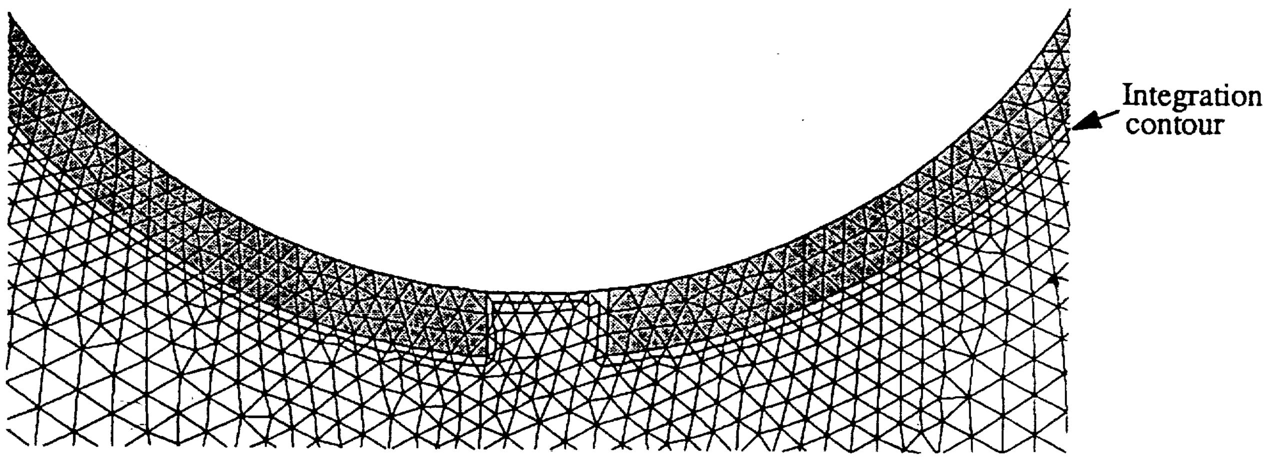


FIGURE 2. Integration contour around a coated conductor with a crack